

**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

1800 WOODFIELD DRIVE, SAVOY, IL 61874-9505  
217-352-1165 FAX 217-352-9307

*Rel date*

*8/8/93*

*SBIR-06.05-1165*

**NASA-CR-194232**

*1/N-3-1-1*

*10314*

*P-338*

**NASA P2T2 INTELLIGENT TRAINER**

**FINAL REPORT**

August 15, 1991

SBIR NAS-9-18170

Welcome to the P2T2 Intelligent Trainer



(NASA-CR-194232) NASA P2T2  
INTELLIGENT TRAINER Final Report  
(Global Information Systems  
Technology) 338 p

N94-70496

Unclas

29/09 0183164



## TABLE OF CONTENTS

SUBJECT	PAGE
Project Summary.....	2
Project Objectives .....	3
Overall Objective.....	3
Supporting Objective .....	3
Work Carried Out.....	4
Overview.....	4
Associated Documentation.....	4
Selecting a Domain and a Simulator .....	6
Knowledge Engineering.....	7
Designing the Part Tasks and Developing the Training Approach .....	8
Part Task Analysis .....	8
Part Tasks.....	9
Training Modes.....	12
Creating a Student Model (Domain Hierarchy) .....	13
Creating an Architecture for the ITS.....	15
Integrating with the Simulator .....	17
Student Feedback and Evaluation.....	17
Editable Parameters .....	18
Demonstrating the System at NASA and Revising as Directed .....	19
Results Obtained.....	20
Estimates of Technical Feasibility .....	22
Appendix A: PDRS Operations Checklist P2T2 Flight Supplement, Generic ProceduresA-1	
Appendix B: Part Task Design Document and User Manual.....	B-1
Appendix C: Programmer's Reference .....	C-1
Appendix D: System Design Overview .....	D-1
Appendix E: System Architecture Design Document.....	E-1
Appendix F: Modifications to the P2T2 .....	F-1
Appendix G: Performance Analysis Design .....	G-1
Appendix H: Error Analysis Design.....	H-1
Appendix I: Student Model Design .....	I-1
Appendix J: Deficiencies Analysis Design.....	J-1

## **PURPOSE OF THE RESEARCH**

The purpose of this research was to integrate an Intelligent Tutoring System (ITS) for automated performance monitoring and evaluation with an existing simulator of NASA's selection.

## **DESCRIPTION OF THE RESEARCH**

An existing kinematic simulator of the shuttle's robotic arm, called the P2T2, was selected, and a part-task training approach based on several tested instructional theories was developed. Twenty-four part tasks were created, each with 3-6 levels of complexity (LOC) and 3-5 trials per LOC. Performance monitoring and evaluation was implemented for efficiency, accuracy, safety, procedural correctness, and camera skills where relevant.

A "Domain Hierarchy" was developed where the Deploy and Retrieve whole tasks were at the top of a tree-like data structure, and supporting knowledges and skills were below. This hierarchy became the basis for the Student Model of the ITS, and is used in skill maintenance.

Two training modes were developed: a "tutorial mode," which is an ordered sequence of part tasks, and a "skill maintenance mode." In skill maintenance mode (based on fault-isolation techniques from electronic troubleshooting), the ITS gives the student one of the two whole tasks as a test, monitors the student unobtrusively, determines suspected problems, and, through propagation of suspicion indices through the Domain Hierarchy and through further testing at lower points in the hierarchy when necessary, arrives at a known weakness. The ITS then tutors the student on the isolated weakness using an appropriate part task(s).

Instructor-editable measures of performance were developed, so that, for example, an instructor can change the degree of efficiency required for a particular task, or allow more time, etc. After each trial, students receive textual and graphic breakdowns of their performance. Student data is kept from one training session to the next.

## **RESEARCH RESULTS**

The results of the research include a methodology for integrating performance monitoring and evaluation with existing simulators, development of the domain hierarchy for student modeling and related instructional information, a part task approach to training a high-performance skill, and a very efficient skill-maintenance methodology.

## **POTENTIAL COMMERCIAL APPLICATIONS**

Potential applications are military, government, and industry currently using simulators. These include military operations, nuclear power plants, airline and aerospace training, and factory automation. Such systems could also become on-line advisors as well as stand-alone (e.g., remote) training devices.



## PROJECT OBJECTIVES

### Overall objective:

Integrate an Intelligent Tutoring System (ITS) for performance monitoring and evaluation with an existing simulator of NASA's selection

### Supporting objectives:

1. Working with NASA training groups, select an existing simulator
2. Determine an existing NASA training lesson relevant to the simulator as a focal point for the ITS
3. Perform knowledge engineering required to implement the ITS
4. Develop the training approach required to monitor and evaluate performance for the simulator (based on Objectives #1, 2 and 3)
5. Use technical approaches, designs, data structures, etc. developed in Phase I where feasible
6. Implement the ITS
7. Demonstrate the ITS to NASA. Revise as directed.

## WORK CARRIED OUT

### OVERVIEW

An existing kinematic simulator of the shuttle's robotic arm, called the P2T2, was selected, and a part-task training approach based on several tested instructional theories was developed. Performance monitoring and evaluation were implemented for efficiency, accuracy, safety, procedural correctness, and camera skills. Two training modes were created: tutorial mode (an ordered sequence of part tasks) and skill maintenance mode (an efficient method of remediating weak skills at some point after initial mastery).

This work was monitored by the Software Technology Branch (Robert T. Savely, Chief), Information Technology Division, Information Systems Directorate (JSC). R. Bowen Loftin was the technical monitor.

### ASSOCIATED DOCUMENTATION

Included in this final report as appendices are detailed supporting design documents for the P2T2 Intelligent Trainer. The body of the report can be seen as an overview or summary of the project and refers to these documents where appropriate. This section gives a brief summary of each document.

- **Appendix A: PDRS Operations Checklist P2T2 Flight Supplement, Generic Procedures**

This document details the procedural steps and related input values for each phase of RMS operation. It was created for students using the P2T2 Intelligent Trainer. The document uses the existing NASA format for flight procedure checklists. Students should use the checklist to know exactly what actions are expected for each task where procedures are relevant.

- **Appendix B: Part Task Design Document and User Manual (August 6, 1991)**

This document gives readers a very detailed description of the overall training philosophy of the P2T2 Intelligent Trainer, various instructional features of the Trainer, how performance monitoring and evaluation is performed, the ordering of tasks in tutorial mode, descriptions of each part task with all its associated levels of complexity and initial performance-measurement values, complete text of the advance organizers, and instructions on changing the editable parameters.

- **Appendix C: Programmer's Reference (August 8, 1991)**

This document was written as a guide to anyone maintaining the P2T2 Intelligent Trainer. (It does not pertain to maintaining either the Silicon Graphics workstation nor the P2T2.) Included are sections on the P2T2 Intelligent Trainer environment, executable files, source code and makefiles, instructions on adding tasks and on integrating with new versions of P2T2, instructions for adding new commands to the interface, and a section on the data files. Also included are appendices which give a listing of the source files, formats of the data files, modifications to the P2T2, and the Domain Hierarchy.

- **Appendix D: System Design Overview (June 6, 1990)**

This document gives an overview of the P2T2 Intelligent Trainer. It serves as a "roadmap" to the more comprehensive subdesigns. The purpose of this document is to enumerate the goals of the Intelligent Trainer, the current constraints imposed by the domain, the high-level data flow of the Intelligent Trainer, and to discuss the philosophical and practical benefits of the selected architecture of the P2T2.IT.

- **Appendix E: System Architecture Design Document (February 7, 1990)**

This document describes the system architecture of the P2T2 Intelligent Trainer. It describes the different operating modes, how they are chosen, and how the student can interact with them. It also describes how the major components of the Intelligent Trainer work together. The major components of the Intelligent Trainer are:

- Performance Analysis
- Error Analysis
- Student Model
- Deficiencies Analysis
- Instructional Assignment (for which see the Part Task Design Document)

- **Appendix F: Modifications to the P2T2 (December 13, 1990)**

This document describes the modifications of the P2T2 as well as other implementation-related topics.

- **Appendix G: Performance Analysis Design (August 21, 1990)**

This document describes the Performance Analysis module of the P2T2 Intelligent Trainer. Performance Analysis conducts an initial analysis of the student's manipulation of the Remote Manipulator System (RMS). Performance Analysis only analyzes the student's current performance and does not look for possible historical causes of the student's behavior. (The historical analysis of the student is left to the Error Analysis and the Student Model modules.)

- **Appendix H: Error Analysis Design (September 13, 1990)**

This document describes the Error Analysis module of the P2T2 Intelligent Trainer. Detailed here is how Error Analysis interprets the evaluation of the student's current performance done by Performance Analysis and determines how the historical record of the student maintained by the Student Model must be changed. Error Analysis seeks to explain in terms of the student's knowledge and skills the errors observed by Performance Analysis.

The document includes sections on how Performance Analysis is carried out and how Error Analysis builds on the results of Performance Analysis; how Error Analysis interprets the student errors located by Performance Analysis; and how the historical model of the student, maintained by the Student Model, should be modified by the Error Analysis module.

- **Appendix I: Student Model Design (August 8, 1990)**

This design document describes the Student Model module of the P2T2 Intelligent Trainer. The role of the Student Model is to represent the relationships between the concepts, knowledge elements and skills of the domain, as well as the state of the student's mastery over the domain (e.g., RMS training) and a history of some aspects of the training for this particular student.

- **Appendix J: Deficiencies Analysis Design (August 17, 1990)**

This document describes the Deficiencies Analysis module of the P2T2 Intelligent Trainer. The document includes a detailed design of how the Deficiency Analysis module carries out its primary functions. The Deficiencies Analysis module, used during skill maintenance, operates on information contained in the Student Model. It decides whether to test for a suspected student weakness or to recommend remediation on a concept, knowledge element, or skill. The recommended test or remediation node is passed to the Instructional Assignment module for further action.

## **SELECTING A DOMAIN AND A SIMULATOR**

Training for the Remote Manipulator System (RMS) was targeted as the testbed for this project for several reasons:

1. RMS training offers a somewhat procedural task. However, it also includes a large skill component.
2. A kinematic simulator of the robotic arm, called the P2T2, hosted on a Silicon Graphics workstation, was available as an interim simulator. This would allow us to make a great deal of progress without impacting the training schedule of the highly-utilized and expensive SMS, SST, or SES simulators. Migration to the SMS or another simulator could wait until the initial system was completed and debugged.

3. The P2T2 trainer exists, but does not currently have an evaluative or instructional component. Creating such a capability for the P2T2 could further off-load training from the SMS, SST, etc.
4. The sponsoring organization, Software Technology Branch, included experts on the P2T2 software, helping minimize impact on RMS instructors.
5. RMS training is vital for current and future missions for both shuttle and space station operations, and increased efficiency and training is required.

## KNOWLEDGE ENGINEERING

Knowledge engineering is the process of finding out how experts perform a task or what specific decisions they make, and what they base their actions and decisions on. This information is then structured and modeled in the developing system. In ITSs, the expert's knowledge or modeled performance is frequently used as a standard for student mastery performance, and the question then becomes, how can the ITS help the student learn both what the expert knows and also how to respond like the expert?

Several methods of performing knowledge engineering were used:

### 1. Interviews

A number of interviews were held with two RMS instructors. (The instructors were generally interviewed separately.) These interviews were held sitting at a table, with Global personnel asking pre-developed questions and the instructor answering. In these interviews, we developed first an overview of the RMS domain: What are the main tasks? What are the supporting tasks? What measures of performance are important? What kind of aids do you currently use in helping students understand the RMS? What kind of support would you as an instructor like from this system? How should the student set up the camera for the various phases of arm operation? How would you evaluate efficiency for a particular task?

Extensive lists of questions were developed and asked to help us understand how NASA trains RMS operators, what training outcomes they expect, and how they evaluate student performance.

### 2. Existing training on simulators, etc.

Global also went to a large number of RMS training sessions on the various simulators (SMS, SST, SES, MDF) and attended an initial RMS classroom training session. During the training sessions, we attempted to answer several questions:

- a. What kind of tasks do the instructors give students during training?
- b. What role do instructors take during this training?
- c. What level of detail do coaching and evaluations take? How are they initiated? What is included in a debriefing?

- d. What kind of questions do students ask, and what kinds of feedback are they looking for?
- e. What kind of training (visual) aids do the instructors use?
- f. What is the domain jargon used in the training?

3. Existing training materials

Global reviewed existing training materials such as the Payload Deployment and Retrieval System Overview Workbook, PDRS Operations Checklist, PDRS Training Curriculum Flow, PDRS Objectives, Space Station RMS Analysis, and various other training analyses. This review helped us further define the RMS domain, determine which objectives could be supported by the P2T2 Intelligent Trainer, find answers to details in the RMS domain without bothering the instructors (whenever possible), and develop the procedural evaluations for the various tasks.

4. Conduct and Analyze "Think-Aloud Protocol" Sessions

Global scheduled four crew members to perform a whole task (deploy the Spartan payload) using the P2T2. These sessions were taped for further analysis. The crew varied in terms of approach and capability. One of them was barely through with initial (classroom) training, one at a medium level of ability, and two of them quite accomplished. During these sessions, the crew were asked to talk about what they were doing and why, and to think out loud as they solved the problem. Global did not coach them in performing the task. Deploying the Spartan turned out to be a good problem, since the grapple fixture is on the opposite side of the payload bay from the arm shoulder. This meant that hitting arm singularities and reach limits was almost inevitable, giving us a deeper look at RMS operations and problems as well as operator styles.

5. Prototype demonstrations and discussions

Global returned to JSC on a number of occasions to give system overview presentations and demonstrations of the work-in-progress to various NASA groups and people. Using the demonstrations, we were able to get specific feedback from the potential users, and to augment or modify the P2T2 Intelligent Trainer while there was ample opportunity (e.g., time remaining).

## DESIGNING THE PART TASKS AND DEVELOPING THE TRAINING APPROACH

### Part Task Analysis

Using the results of the knowledge engineering discussed above, an initial listing and brief description of all potential RMS whole and part tasks were developed by Dr. J. Wesley Regian. Actual tasks were then designed by Global, including an initial set of performance measures for each task. (Dr. Debra Johnson contributed helpful input on the initial performance measures.) Only nominal sub-tasks were included in the part-task analysis (i.e., dealing with possible problems with the arm, such as a runaway arm, were not addressed), since the P2T2 does not model faults. Each task design also included

an associated advance organizer, initial camera views, levels of complexity (increasing difficulty, such as using larger payloads), and associated visual aids. In addition, Global designed the surrounding training system, which includes such things as login/logoff procedures, a performance monitor (which shows the student his or her progress within the specific task related to mastery), a timer, feedback and real-time hints, database for student performance data, editable parameters, student progress tracking within the part task sequence, etc.

Complete task and training system descriptions can be found in the *Part Task Design Document and User Manual*, Appendix B.

### Part Tasks

A large number of part tasks to support deploying and retrieving payloads were designed and implemented. Each task uses the P2T2 simulator with most or all of its inherent capabilities intact. The student must then perform some discreet task, like grapple a payload.

A part task is composed of three to six Levels of Complexity (LOCs), each of which is composed of a number of trials (generally three to five). The student must first perform the task a number of times in its easiest manifestation; each of these is called a trial. For example, smaller payloads are easier to maneuver than larger payloads, so the first LOC for the Loaded Arm Phasing part task would require moving a small payload to a particular position and attitude. If the task has three trials per LOC, then the student would get three similar problems to solve. The next LOC might involve moving a larger payload a longer distance, and the final LOC might have the student move large payloads to locations that require avoiding reach limits and singularities. Within each LOC, the student would have the same relative problem to solve per trial, but the arm may start in a slightly different location, the specific payload may change though remain the same relative size, or the goal position may change.

After each trial, the student receives a textual evaluation. Depending on what was measured in the part task, the student gets an overall score and judgement (e.g., Passed) as well as a breakdown of each measure. Figure 1 shows an evaluation for a complex task for which all possible measures are evaluated. Performance data is kept in a simple database.

Also accessible after each trial is a graphical depiction of student performance. The student can see his or her best performance for the part task, average of all performances, and the most current performance. See Figure 2.

You did not pass the trial.

Your final score is 49.9

Accuracy ( 15% of final): 0.0 (Failed)  
Ending position : -806.2 -13.3 -862.3 (PLID 0)  
Goal position : -806.1 -81.0 -862.3 (PLID 0)

Safety ( 35% of final): 100.0 (Passed)  
You did not have any safety violations during the trial.

Efficiency ( 10% of final): 89.4 (Passed)  
Time evaluation ( 75% of efficiency): 100.0  
4 seconds were used out of 3 minutes allowed

Path evaluation ( 25% of efficiency): 97.8

Procedure evaluation ( 35% of final): 0.0 (Failed)  
You started moving to post-release position before you set EE mode to AUTO.  
You did not depress the EE release switch when attempting to release the payload.  
You did not set EE mode to AUTO when attempting to release the payload.  
You did not finish moving to post-release position.  
You did not finish releasing the payload.  
You did not set EE mode to off after moving to post-release position.  
You did not engage the RMS brakes after moving to post-release position.  
You did not switch the joint dial to CRIT TEMP after moving to post-release position.  
You did not set SPEC94 ITEM 3 PLID to 0 after moving to post-release position.  
You did not set the SPEC94 ITEM 24 PL INIT ID to 0 after moving to post-release position.

Camera evaluation ( 5% of final): 100.0 (Passed)  
You used the EE cam correctly 100% of the time during the trial. Acceptable usage is 90% of the time.  
You had a good view of the EE and the Grapple Fixture 100% of the time. Acceptable usage is 80% of the time.

Figure 1: Textual Performance Evaluation



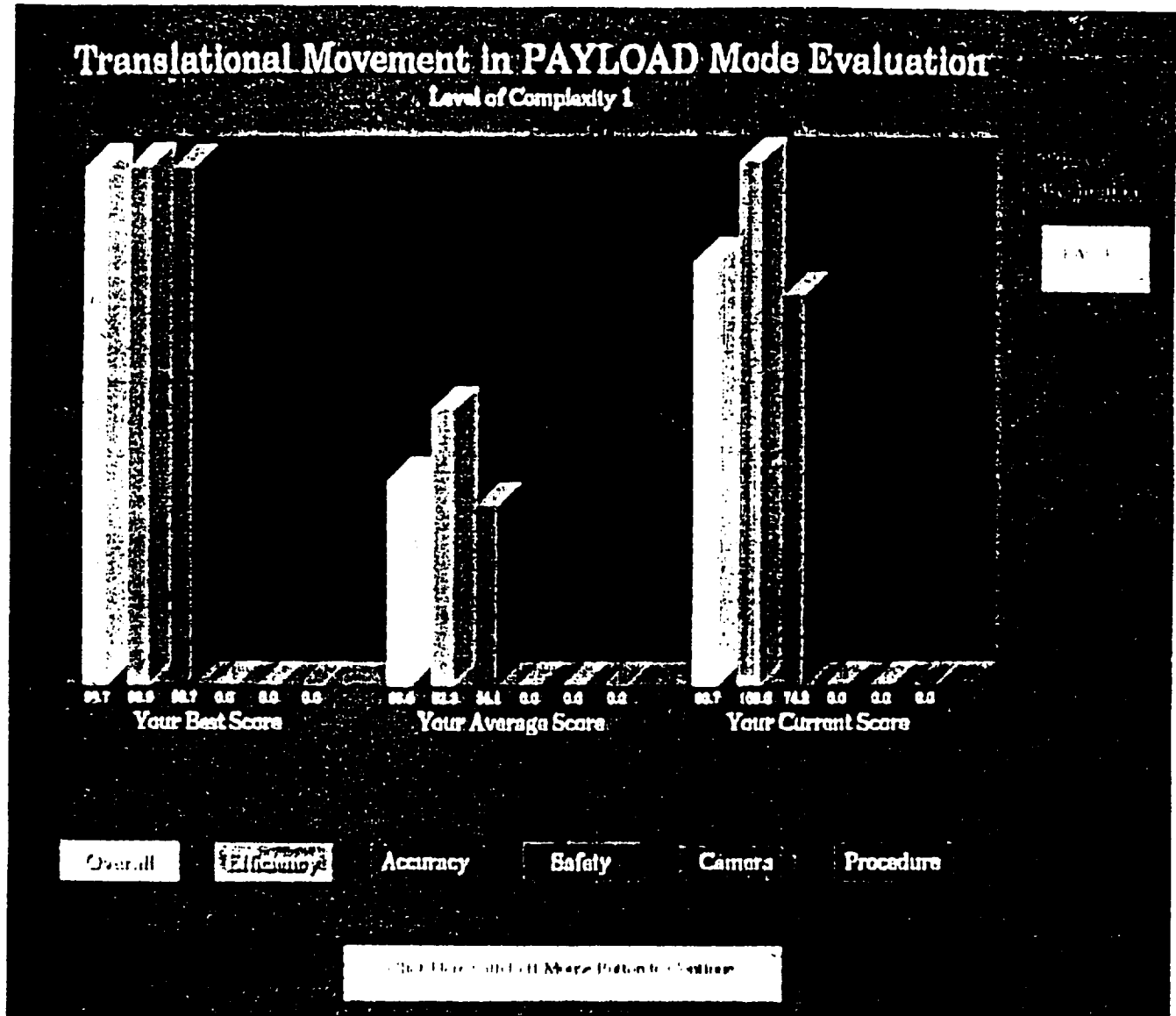


Figure 2: Graphical Performance Chart

ORIGINAL PAGE IS  
 OF POOR QUALITY

Each part task may also have various visual aids available for the student. For example, a "godseye" view may be used. (Godseye views are non-realistic views of the orbiter and payload bay, as though seen from outer space.) Such visual aids are always faded after one or two LOCs so that the student does not come to rely on them to solve the general problem. However, they are sometimes still available to the student in the form of "limited resource buttons." For example, the student may be allowed to press a graphical button (using the mouse) to receive a godseye view for 30 seconds; and three such godseye resource buttons may be available during the remaining portion of the part task. (The number is editable by instructors.)

Some tasks have real-time help available. For example, the student may be in course rate while going in for a grapple instead of the slower Vernier rate. A beep will sound, and the HELP button becomes active (i.e., white instead of stippled); in addition, the text "New HELP message" appears. The student can then press the HELP button and be told to "Use Vernier rate within 10 feet of orbiter structure." As soon as the student switches to Vernier rate (whether or not s/he has read the help message), the button becomes inactive.

### **Training Modes**

Two training modes were developed besides the existing (unmonitored) simulation mode: "Tutorial Mode" and "Skill Maintenance Mode."

Tutorial Mode is an ordered sequence of part tasks, each of which the student is expected to master before proceeding to the next part task. The preceding section on Part Tasks gives a good idea of how part tasks function during Tutorial Mode. For more information, see Appendix B: Part Task Design Document and User Manual.

Skill Maintenance Mode, which follows Tutorial Mode, is an efficient method of remediating the student's skills when they start to decay. It uses a test/tutor paradigm, testing wherever possible and tutoring only when necessary, since tutoring, with its associated levels of complexity (LOCs), trials per LOC, coaching, hints, etc., takes longer than testing.

Skill Maintenance Mode seeks first to determine weaknesses in the student's understanding of the domain by testing. The ITS first gives the student one of the whole tasks (Deploy or Retrieve) as a test, monitors the student unobtrusively, determines weaknesses or suspected problems, and, through further testing at lower points in the Domain Hierarchy (see the next section) when necessary, arrives at a known weakness. It then tutors the student using an appropriate part task at a median level of difficulty (LOC). If the student fails at the current LOC three times consecutively, s/he goes back one LOC. If the student is already at LOC 1 and fails three times consecutively, the particular part task is terminated with a message to the student to see his/her instructor. Otherwise, the part task works the same way as it did in Tutorial Mode.

Skill Maintenance mode uses concepts and algorithms from electronic troubleshooting (fault isolation diagnosis), where the Domain Hierarchy is analogous to an electronic circuit. Since tutoring takes much longer than testing, our approach allows a very efficient method of remediating decayed knowledge and skills. For more information,

see Appendix G: Performance Analysis Design; Appendix H: Error Analysis Design; and, especially, Appendix J: Deficiencies Analysis Design.

### **Creating a Student Model (Domain Hierarchy)**

A "Domain Hierarchy" was developed where the Deploy and Retrieve whole tasks were at the top of a tree-like data structure, and all supporting knowledge and skills were below (see Figure 3). This hierarchy became the basis for the Student Model of the ITS, and is used in the Skill Maintenance Mode.

Attached to each node are:

- Instructional information (date last remediated, date mastered, etc.)
- Skill maintenance status (e.g., mastered, unknown, misused, suspect, etc.)
- A part task for testing and/or remediation

Special links were created between the nodes to indicate types of dependencies and how to deal with them during fault-isolation propagation. For example, we know that if a student gets a poor efficiency rating while performing the Grapple part task, then the student has problems using the translational hand controller (THC). We know this because use of the hand controllers is the only determinant of efficiency (besides time) during the Grapple part task, and because the THC is the only hand controller required to perform that task due to the way the arm is set up at the beginning of each trial (i.e., it is in the pre-grapple position and attitude). Thus, when values are propagated through the Domain Hierarchy during Skill Maintenance Mode, only the THC node is updated for a poor efficiency rating during the Grapple task.

See Figure 3 for a graphical depiction of the Domain Hierarchy.

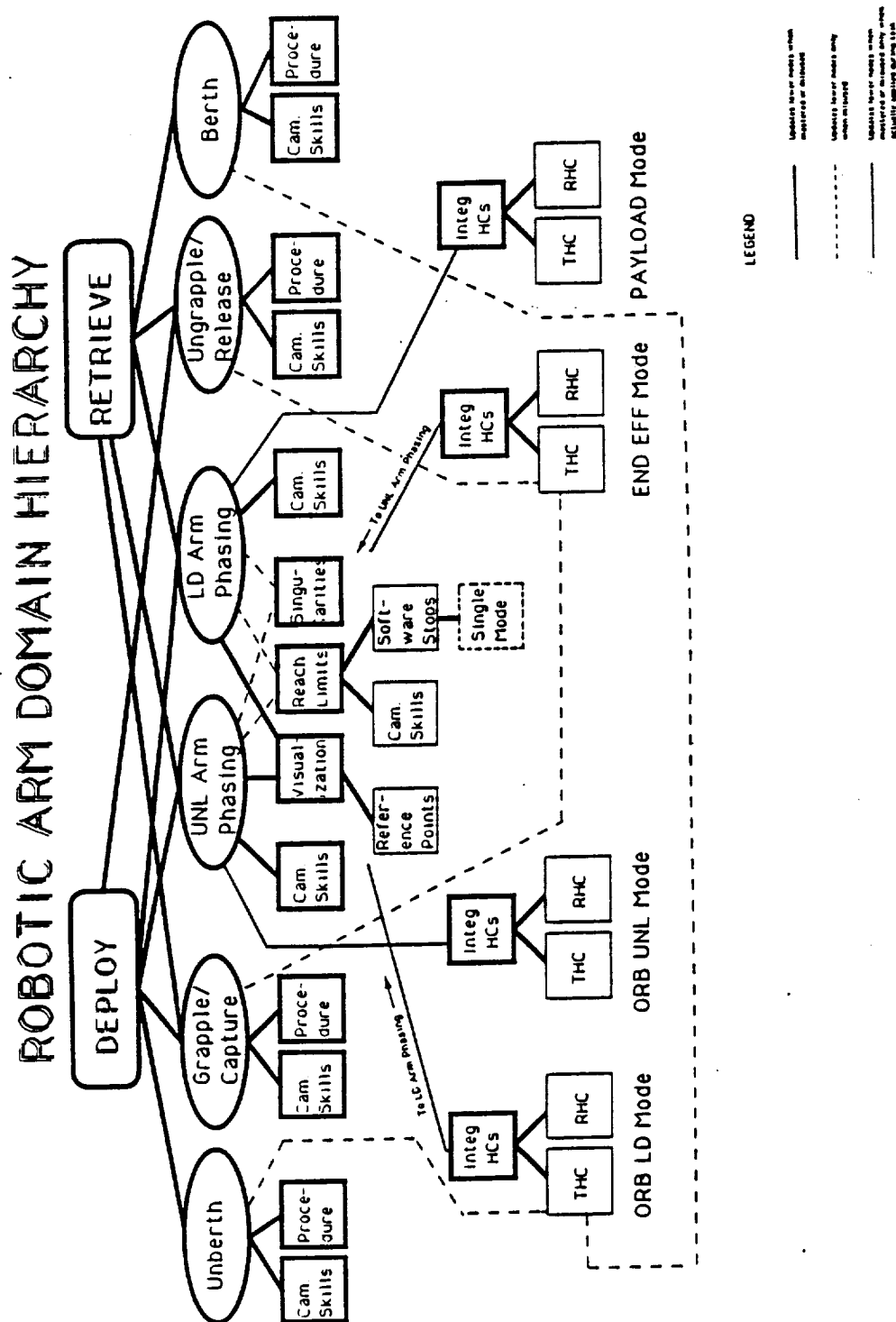


Figure 3: RMS Domain Hierarchy

## CREATING AN ARCHITECTURE FOR THE ITS

In developing the system architecture, Global initially reviewed the ITS architecture we developed during Phase I of this project. We also reviewed the architecture and methods of an immediately-previous ITS we developed for the Army. Any work we could leverage from those projects to solve requirements in this project would allow us to go farther in this project.

From the NASA Phase I project, we took the procedural network concept, which allows us to monitor and evaluate performance of a procedure flexibly. For example, a step in the procedure may not have to be performed until a particular time (as opposed to "After Step 2"), whereas the next five steps may require a rigid sequence of actions from the student. Another situation where the procedural network is important is when a student performs an incorrect step, then changes it later. The network allows this sort of flexible representation of procedures.

From the Army project, we borrowed the basic system architecture (e.g., how the student model, expert model, deficiencies analysis, etc., work together), and then modified it for the unique aspects of RMS and of part-task training (see Figure 4). The RMS part-task training required a more structured approach to skill maintenance, for example, whereas in the Army domain (HAWK missile system tactical operations), the subscenarios for testing and remediation were created totally "on the fly," and not pre-designed part tasks. Also, we re-used almost the entire fault-isolation approach and algorithms to skill maintenance without change from the Army project. Finally, we borrowed the Domain Hierarchy architecture from the Army project, although it required extensive changes for both the new domain and also for accommodating the part-task approach of the RMS training.

Next, Global analyzed the P2T2's Silicon Graphics environment to decide whether to interface an 80386 machine to it for the P2T2 Intelligent Trainer, or to run the ITS in parallel within the Silicon Graphics. We selected the latter approach for simplicity and cost.

The final functional architecture of the P2T2 Intelligent Trainer is shown in Figure 4.

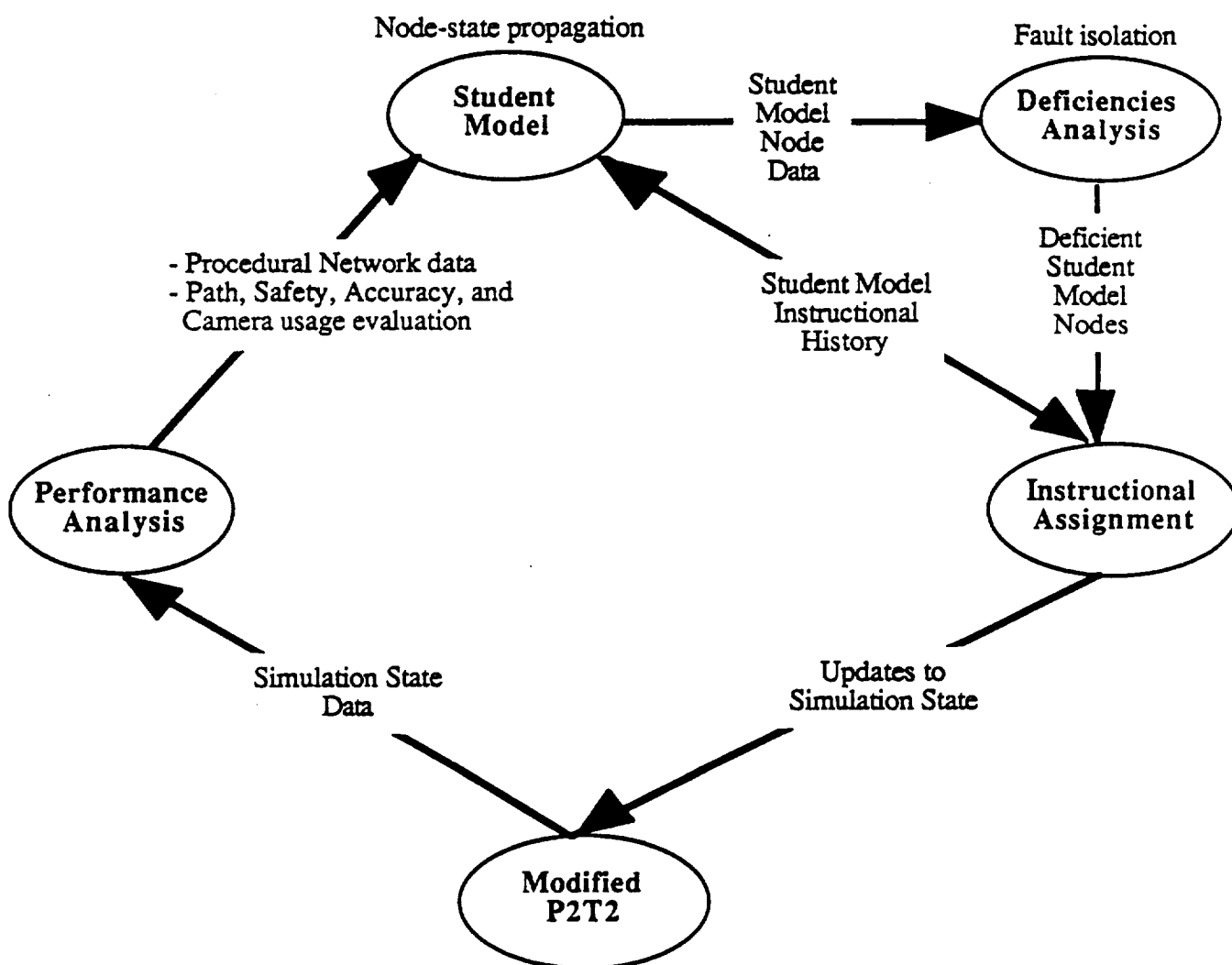


Figure 4: Functional Architecture

For more information on the system architecture and the specific functioning of the various modules, see the following design documents:

- System Design Overview
- System Architecture Design Document
- Deficiencies Analysis Design
- Performance Analysis Design
- Error Analysis Design

## INTEGRATING WITH THE SIMULATOR

A method of integrating with the P2T2 simulator was selected that required the fewest possible changes to the P2T2 code. We also decided to share processing resources with the P2T2 on the Silicon Graphics workstation (as opposed to, for example, interfacing an 80386 workstation to the Silicon Graphics machine to do the performance monitoring and evaluation and other ITS-related tasks).

The following modifications and extensions were made to the P2T2 in order to develop the P2T2 Intelligent Trainer for part-task training:

- Load new object models into the simulation (such as visual aids)
- Display and hide new object models
- Load a new simulation scenario (payload, RMS configuration, visual aids, and initial simulation state)
- Pause a simulation scenario to provide instructional feedback
- Stop a simulation scenario
- Reset a simulation scenario to some previous point
- Alter the simulation controls (disable the RHC, THC, keyboard, buttons, dials, and mouse)
- Provide simulation state data (RMS position and attitude, mode and rate settings, position and attitude of grapple fixtures, payloads and other objects, etc.) to the IT

See also Appendix F: Modifications to the P2T2 and Appendix D: System Design Overview.

## STUDENT FEEDBACK AND EVALUATION

A part task is composed of a number of Levels of Complexity (LOCs), each of which is composed of a number of trials. These are defined in the *Part Task Design Document and User Manual* (Appendix B) per task. (See also the previous section, "Part Tasks" for an overview.)

At the end of each trial of a part task, students receive a textual breakdown of their performance. Performance measures include efficiency, accuracy, safety, procedural correctness, and camera usage where relevant. Each measure details what percentage it contributes to the overall score, what the specific score was for that measure, and details of whatever sub-measures were used (e.g., both path traveled and

time used are sub-parts of the efficiency measure for most tasks). Also included in the textual evaluation are any procedural steps that were omitted; specific reasons why any safety points were deducted; how the cameras should have been adjusted (if they weren't); etc. An evaluation graph is also available at the end of each trial that sums up information over the entire part task, including best, average, and most-current performance. Student data is kept from one training session to the next.

See Figures 1 and 2 for examples of the textual and graphical evaluations.

Scoring a student's performance is a complex process. For example, what is efficient enough? How important is efficiency compared to camera usage, or procedural correctness? Note that all aspects of scoring and evaluation are editable by instructors. See both Appendix A: *Part Task Design Document and User Manual* (especially Appendices A and B of that document), and also the section below, "Editable Parameters."

## EDITABLE PARAMETERS

Two aspects of trainer functioning were identified that led to the development of "editable parameters."

1. During the knowledge engineering phase of this project, it became obvious that shuttle training and space station training had different philosophies concerning student performance evaluation. For example, shuttle trainers were far more concerned with safe operation than with efficiency (when efficiency includes a time element), and therefore did not want time as an element of the efficiency measure. Space station, on the other hand, wanted to quantify student performance and to "wash out" less able operators early in the training; thus, time as a measure of efficiency was important, since the ability to perform a task quickly is frequently an indication of ability.
2. Another unrelated but important aspect of performance measurement that was problematic is indicated by such questions as, How efficient is efficient enough? How much time is required to do this task? How many points should be deducted for safety violations? How close must the arm be to the goal position for sufficient accuracy? What percentage of the time must a student use both hand controllers simultaneously? No one can really know the answers to these questions in advance of watching students actually use the resultant trainer.

To handle these two problems, Global created editable parameters.

Editable parameters are values that the P2T2 Intelligent Trainer uses to quantify performance measures. Also, editable parameters determine various aspects of how the trainer functions; for example, whether the timer is visible to the student. There are two types of editable parameters (see also below): global, which affect all tasks when changed; and task-specific.



Editable parameters can be changed by instructors. Both the global and task-specific parameters are kept in separate, well-documented files so that they are easily accessible and understandable. (Appendices A and B of the *Part Task Design Document and User Manual*, itself appendix B of this document, detail what parameters can be changed and how to perform this task.)

All mastery and advancement criteria, as well as a large number of other aspects of the P2T2 Intelligent Trainer, are changeable by instructors. An example of an editable parameter is that, instead of three consecutive error-free performances as currently specified, the instructor can change the value to five. Number of trials that must be mastered per LOC, time limits, and visual-aid buttons (as well as length of time the visual aid returns) are just a few of the large quantities of parameters that are editable.

There are two types of editable parameters: global and task-specific.

Global parameters: A global parameter affects all tasks. An example is use of the timer, which can be "turned off" at the global level and is not then used in any task. This saves instructors from having to turn the timer off for every part task. Global parameters are read up by the P2T2 Intelligent Trainer when the student signs on.

Task-specific parameters: A task-specific parameter affects only that particular part task. An example of such a parameter is the amount of time allowed for a student to do the particular task. Every part task and each of the two whole tasks have their own editable parameters file. Task-specific parameters are read up by the P2T2 Intelligent Trainer before each task.

## DEMONSTRATING THE SYSTEM AT NASA AND REVISING AS DIRECTED

A significant number of trips were made to NASA during the two-year duration of this work to demonstrate progress and functionality to the various user groups and to get customer feedback. In addition, project overviews were generally available for people who weren't familiar with the project. People at all levels of Shuttle and Space Station training were invited to attend the presentations and demonstrations. Changes desired by our user groups were incorporated whenever possible.

The last six weeks of the project were spent entirely at the guidance of Space Station training, since they emerged as the most likely user of the P2T2 Intelligent Trainer. Global created an initial list of possible changes based on our own observations at NASA (we did not have the hand controller chair in Illinois), the comments of Dr. J. Wesley Regan, and a list from the Space Station instructional specialist.

## RESULTS OBTAINED

### Overview

Although a controlled study of the efficiency of the P2T2 Intelligent Trainer will not be completed by NASA until November, 1991, the reaction from various users at NASA has been quite enthusiastic. We believe this trainer is a viable training device for initial shuttle and space station training. Possibly the most exciting results will be efficiency in skill maintenance, expected to yield results of at least 50% less training time for remediation of decayed knowledge and skills. However, the many hours of controlled performance monitoring and diagnosis and real-time feedback an operator will receive using the Tutorial Mode (an ordered sequence of part tasks) of the Trainer will have an appreciable impact on training operations. Coupled with the fact that the P2T2 Intelligent Trainer is a stand-alone training device, this also means a significant increase in newly-available instructor time to deal with more complex or higher-level training requirements.

### Background: The P2T2 Simulator as a Training Device

A study of the P2T2 itself, performed by NASA in 1989, concluded that, first of all, "facility time in existing PDRS facilities [e.g., SMS, SES, SST, MDF] [are] both scarce and expensive," and finally that the P2T2 simulator provided "an easily accessible and reconfigurable RMS nominal operations trainer." System effectiveness was rated across eight measures, and the lowest measure (for effectiveness of panel A8 visual implementation, since resolved) was nevertheless 3.5 (where 3.0 meant "acceptable training value"). The results were the same (i.e., 3.5 was the lowest score) across seven task-specific effectiveness measures.

This study analyzed the PDRS training flow for CB and DF training requirements, and a significant number of training objectives were identified as suitable for training on the P2T2 simulator. Most of the RMS operations targeted in this NASA analysis for delivery on the P2T2 exist as part tasks in the P2T2 Intelligent Trainer.

### The P2T2 Intelligent Trainer Works

The primary product of this contract, the P2T2 Intelligent Trainer, is a full-functioning, stand-alone trainer for RMS nominal operations. Current product functionality includes: a training-specific user interface ("tutoring window") integrated with the P2T2 interface; automatic performance monitoring and evaluation; student login, data collection, record keeping, lesson sequencing, and logoff functionality; real-time coaching and hints; automatic presentation of part tasks based on student diagnosis; mastery training across the RMS domain in a principled sequence of part tasks; and a skill-maintenance mode for follow-on training and remediation. The system is quite easy to start up, and automatically tracks the student and presents new tasks: once the student knows how to use the P2T2 simulator, the P2T2 Intelligent Trainer requires little extra training (perhaps only instruction on how and when to use the mouse to make certain selections or responses).

### **Initial Response**

The initial response from most of the potential user groups, as well as the numerous other NASA personnel who have tried out the system, has been very enthusiastic. In addition, Space Station training has invested a large amount of time in evaluating the trainer and specifying changes, most of which were implemented (as time allowed).

## ESTIMATES OF TECHNICAL FEASIBILITY

This project has demonstrated conclusively that a stand-alone training device can be created by integrating automated performance monitoring and evaluation with an existing simulator. We see no reason that such work could not also be done using one of the higher-level simulators such as the Single System Trainer (SST). In such a case, we would have additional problems to solve, such as receiving and processing large amounts of data from the simulator. However, Global designed a data filter architecture for such a task during Phase I of this project.

The biggest problem of the P2T2 Intelligent Trainer, which is likewise a problem with the P2T2 itself, is computational speed. The simulation graphics are modeled at a very fine level of detail, and this means the Silicon Graphics workstation cannot always run the simulation in real time. (Using a Silicon Graphics top-of-the-line VGX machine, the P2T2 Intelligent Trainer attained 33% real-time functioning when both hand controllers were simultaneously pushed to maximum displacement. Most of the time, however, the system ran at 80-100% real time. These numbers are probably not much different for the P2T2 itself, although that was not tested.) If real-time operation is important, the P2T2 allows users to switch to wireframe graphics; in this mode, the simulation runs smoothly. Also, the graphic display ratio could be changed to displaying one out of every two or three frames, a slightly less smooth simulation, but probably adequate.

No other system resources, such as RAM, disk space, etc., created boundaries on the viability of the P2T2 Intelligent Trainer. Also, the number of expert system rules, which occasionally impact the functioning of an expert system, were kept at a manageable level. We feel that use of C and C++ contributed to the successful functioning of the end product.

### Conclusion

The results of this project demonstrate clearly that the P2T2 can function as a stand-alone training device that does NOT require instructors to stand by. In addition, the trainer has certain objective training measures, so students will get consistent training to mastery by a very patient device. The workstation-based trainer was less expensive to develop and maintain than a full-up realistic simulator such as the SST. This stand-alone trainer allows instructors, already extremely busy, to use their time more profitably for answering complicated questions, training malfunctions, and training higher skills. It also allows the system to be fielded at remote sites (e.g., on the space station) when desired.

**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix A: PDRS Operations Checklist P2T2 Flight Supplement, Generic Procedures**

---

<p><b>APPENDIX A: PDRS OPERATIONS CHECKLIST P2T2 FLIGHT SUPPLEMENT, GENERIC PROCEDURES</b></p>
--



FLIGHT DATA FILE

# PDRS Operations Checklist

P2T2 Flight Supplement

---

Generic Procedures

[ Note: This PDRS Operations Checklist is specifically designed for training using the P2T2 Intelligent Trainer. However, it is intended to be as similar as possible to actual checklists that you will use for specific missions. Therefore, information given in this training supplement that does not normally appear in official operations checklists is placed in brackets. ]



<u>CONTENTS</u>	<u>PAGE</u>
<u>NOMINAL DEPLOY OPS</u> .....	1-1
GRAPPLE (Generic) .....	1-1
UNBERTH (Generic) .....	1-3
RELEASE (SPAS) .....	1-5
<u>NOMINAL RETRIEVE OPS</u> .....	1-8
POISE FOR CAPTURE (SPAS) .....	1-8
CAPTURE (SPAS) .....	1-9
BERTH (Generic) .....	1-11
UNGRAPPLE (Generic) .....	1-14

NOMINAL DEPLOY OPS

PAGE

GRAPPLE.....	1-1
SETUP .....	1-1
MNVR TO PRE-GRAPPLE POSITION .....	1-1
GRAPPLE .....	1-2
UNBERTH.....	1-3
SETUP .....	1-3
MNVR TO LOW HOVER POSITION .....	1-3
MNVR TO RELEASE POSITION .....	1-4
RELEASE.....	1-5
SETUP .....	1-5
RELEASE .....	1-5
[MNVR TO PRE-CRADLE POSITION .....	1-7]

<u>NOMINAL RETRIEVE OPS (SPAS)</u>	<u>PAGE</u>
POISE FOR CAPTURE.....	1-8
SETUP .....	1-8
POISE FOR CAPTURE .....	1-8
CAPTURE.....	1-9
SETUP .....	1-9
CAPTURE .....	1-9
BERTH.....	1-11
SETUP .....	1-11
MNVR TO LOW HOVER .....	1-11
BERTH .....	1-12
PREP FOR UNGRAPPLE .....	1-14
UNGRAPPLE .....	1-14
[MNVR TO PRE-CRADLE POSITION .....	1-15]



## GENERIC GRAPPLE

### 1. SETUP

A7U

CCTV - Config for grapple  
- RMS Wrist, zoom out

SM 94 PDRS CONTROL
--------------------

✓PL ID - ITEM 3 - 0 EXEC  
✓INIT ID - ITEM 24 - 0 EXEC

### 2. MNVR TO PRE-GRAPPLE POSITION

RATE - as reqd (VERN within 10 ft)  
BRAKES - OFF (tb-OFF)  
MODE - ORB UNL, ENTER

Mnvr to Pre-grapple position

[Pre-grapple position is defined as having the EE approximately 5 feet from the grapple fixture, lined up with target. In this case, pre-grapple has the following coordinates:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-908	13.7	<del>-494.7</del>	270	0	180.5	0
SY	SP	EP	WP	WY	WR	

BRAKES - ON (tb-ON)  
MODE - not DIRECT  
JOINT - CRIT TEMP



### 3. GRAPPLE

A7U

✓CCTV - Config for grapple

On MCC Go for grapple,  
DAP: VRCS or free drift  
[assume you have rec'd]

RATE - VERN (RATE MIN tb-ON)  
BRAKES - OFF (tb-OFF)  
MODE - END EFF, ENTER

Mnvr to grapple envelope

CAUTION  
Monitor EE tb timing to  
prevent EE motor burnout

EE MODE - AUTO

EE CAPTURE sw - depress (mom)

<p>✓ <input type="checkbox"/> RIGID</p> <p><input checked="" type="checkbox"/> DERIGID</p>	<p><input type="checkbox"/> CLOSE</p> <p><input checked="" type="checkbox"/> OPEN</p>	<p><input type="checkbox"/> CAPTURE</p> <p><input checked="" type="checkbox"/> EXTEND</p>	<p><u>CRITICAL TIMES (28 sec total):</u> CAPTURE tb - gray, then CLOSE tb - gray, 3 sec max, then RIGID tb - gray, 25 sec max</p>
--	---	---	---

EE MODE - OFF  
BRAKES - ON (tb-ON)

DAP: as reqd

SM 94 PDRS CONTROL

PL ID - ITEM 3 +1 EXEC  
INIT ID - ITEM 24 +1 EXEC

Record POS/ATT and JOINT ANGLES

X	Y	Z	PITCH	YAW	ROLL	PL ID
SY	SP	EP	WP	WY	WR	

## [GENERIC] UNBERTH

[Note: "Low Hover" is a specific position where the payload is directly over the V-guides and Z = -650. Therefore, this value is both generic and payload-specific and is defined below for each payload you may have to unberth during P2T2 training.]

### 1. SETUP

Review LOADED (Cue Card, P2T2-LOADED/RELEASE)

### 2. MNVR TO LOW HOVER POSITION

A7U

CCTV - Config for Mnvr to Low Hover

✓RATE - VERN (RATE MIN tb-on)  
BRAKES - OFF (tb-OFF)  
MODE - ORB LD, ENTER

Mnvr payload to Z = -650 (LOW HOVER)

SPASI:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-895.0	0.0	-650.0	360.0	360.0	360.0	1
SY	SP	EP	WP	WY	WR	
-42.1	83.7	-88.1	-98.9	14.3	-116.3	

GRO:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1089.5	0.6	-650.0	0.3	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
-18.5	74.8	-68.4	-93.0	-10.0	-51.8	

HST:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1010.4	-90.9	-650.0	360.0	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
-37.4	92.2	-85.2	-109.2	15.4	-121.5	



LDEF:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-892.1	-14.2	-650.0	0.4	359.7	90.0	1
SY	SP	EP	WP	WY	WR	
-19.2	97.5	-93.3	-78.5	-37.8	-135.9	

SPARTANH:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-712.1	-0.6	-550.0	359.7	359.6	360.0	1
SY	SP	EP	WP	WY	WR	
-81.3	93.0	-119.3	-112.8	6.6	193.8	

IBSS:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-890.4	-94.7	-650.0	0.0	270.0	0.0	1
SY	SP	EP	WP	WY	WR	
-21.8	86.1	-82.7	-95.0	4.2	41.3	

BRAKES - ON (tb-on)

3. MNVR TO RELEASE POSITION [SPAS]

A7U

CCTV - Config for Mnvr to Rel position

DAP: free drift

RATE - as reqd (VERN within 10 ft)

BRAKES - OFF (tb-OFF)

ORB LD to Release position:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-820	0	-850	90	0	90	1
SY	SP	EP	WP	WY	WR	
-34.7	+119.4	-118.6	+5.6	-0.3	-96.9	

BRAKES - ON (tb-ON)

MODE - not DIRECT

JOINT - CRIT TEMP

## SPAS RELEASE

### 1. SETUP

Review RELEASE (Cue Card, P2T2-LOADED/RELEASE)

✓POS/ATT and JOINT ANGLES [SPAS]

X	Y	Z	PITCH	YAW	ROLL	PL ID
-820	0	-850	90	0	90	1
SY	SP	EP	WP	WY	WR	
-34.7	+119.4	-118.6	+5.6	-0.3	-96.9	

✓SAFING tb - gray  
PARAM sel - JOINT ANGLE

A7U

CCTV - RMS Wrist, zoom out

### 2. RELEASE

On MCC Go for Release

✓RATE - VERN (RATE MIN tb-on)  
✓BRAKES - OFF (tb-OFF), unless DIRECT/BACKUP  
MODE - END EFF, ENTER

DAP: free drift

EE MODE - AUTO







EE RELEASE sw - depress (mom)

[ When OPEN tb - gray:

Mnvr arm clear of GF, payload to:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-806.1	-81.0	-862.3	268.5	270.0	89.0	

Note: It is more important to pull *straight back* on the arm to about 5 ft. in relation to the GF than to achieve these coordinates. ]

	RIGID	CLOSE	CAPTURE	<u>CRITICAL TIMES (28 sec total):</u>
✓				
	DERIGID	OPEN	EXTEND	
				

DERIGID tb - gray, 5 sec max,  
then  
OPEN tb - gray, 3 sec max,  
then  
EXTEND tb - gray, 20 sec max

BRAKES - ON (tb-ON)

EE MODE - OFF

MODE - not DIRECT  
JOINT - CRIT TEMP

[ 3. MNVR TO PRE-CRADLE POSITION

A7U

CCTV - Config for Mnvr to Pre-Cradle

SM 94 PDRS CONTROL
--------------------

PL ID - ITEM 3 +0 EXEC  
INIT ID - ITEM 24 +0 EXEC

✓RATE - as reqd (VERN within 10 ft)  
✓BRAKES - OFF (tb-OFF)  
MODE - ORB UNL, ENTER

SPASI:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1261.2	-145.7	-551.4	4.7	1.7	359.9	0
SY	SP	EP	WP	WY	WR	
0.0	25.0	-25.0	5.0	0.0	0.0	

BRAKES - ON (tb-ON)  
MODE - not DIRECT  
JOINT - CRIT TEMP

## SPAS POISE FOR CAPTURE

### 1. SETUP

A7U

Config CCTVs as reqd

SM 94 PDRS CONTROL
--------------------

✓ PL ID - ITEM 3 +0 EXEC  
✓ INIT ID - ITEM 24 +0 EXEC

### 2. POISE FOR CAPTURE

RATE - as reqd

ORB UNL to poise for capture (✓ ITEMS 18-25)

X	Y	Z	PITCH	YAW	ROLL	PL ID
-860.0	-35.0	-768.0	44.0	291.0	226.5	0
SY	SP	EP	WP	WY	WR	
-36.5	111.2	-102.0	-14.7	-38.3	195.4	

BRAKES - ON (tb-ON)  
✓MODE - not DIRECT  
JOINT - CRIT TEMP

## SPAS CAPTURE

### 1. SETUP

✓Target overlays attached as needed

SM 94 PDRS CONTROL

✓PL ID - ITEM 3 - 0  
✓INIT ID - ITEM 24 - 0

Review RMS-CAPTURE/LOADED (Cue Card)

### 2. CAPTURE

CAUTION  
Monitor EE tb timing to  
prevent EE motor burnout

A7U

CCTV, RMS/Wrist - zoom out

RATE - VERN (RATE MIN tb-ON)  
BRAKES - OFF (tb-OFF)  
MODE - END EFF, ENTER

When grapple fixture in view and stable,  
DAP: free drift

EE MODE - AUTO  
Mnvr to GF  
EE CAPTURE sw - depress (mom)

✓ RIGID CLOSE CAPTURE  
DERIGID OPEN EXTEND

#### CRITICAL TIMES (28 sec total):

CAPTURE tb - gray, then  
CLOSE tb - gray, 3 sec max,  
then  
RIGID tb - gray, 25 sec max

EE MODE - OFF  
BRAKES - ON (tb-ON)  
✓ MODE - not DIRECT  
JOINT - CRIT TEMP

SM 94 PDRS CONTROL
--------------------

PL ID - ITEM 3 +1 EXEC  
INIT ID - ITEM 24 +1 EXEC

[GENERIC] BERTH

1. SETUP

DAP: B/AUTO/VERN

SM 94 PDRS CONTROL

/ PL ID - ITEM 3 +1  
/ INIT ID - ITEM 24 +1

CCTV - Point CCTV as reqd toward EE/GF interface  
for view of uncmd derig/rel

2. MNVR TO LOW HOVER

RATE - as reqd

ORB LD to Low Hover (/ ITEMS 18-25):

SPASI:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-895.0	0.0	-650.0	360.0	360.0	360.0	1
SY	SP	EP	WP	WY	WR	
-42.1	83.7	-88.1	-98.9	14.3	-116.3	

GRO:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1089.5	0.6	-650.0	0.3	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
-18.5	74.8	-68.4	-93.0	-10.0	-51.8	

HST:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1010.4	-90.9	-650.0	360.0	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
-37.4	92.2	-85.2	-109.2	15.4	-121.5	



LDEF:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-892.1	-14.2	-650.0	0.4	359.7	90.0	1
SY	SP	EP	WP	WY	WR	
-19.2	97.5	-93.3	-78.5	-37.8	-135.9	

SPARTANH:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-712.1	-0.6	-550.0	359.7	359.6	360.0	1
SY	SP	EP	WP	WY	WR	
-81.3	93.0	-119.3	-112.8	6.6	193.8	

IBSS:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-890.4	-94.7	-650.0	0.0	270.0	0.0	1
SY	SP	EP	WP	WY	WR	
-21.8	86.1	-82.7	-95.0	4.2	41.3	

A8U

BRAKES - ON (tb-ON)

### 3. BERTH

CCTV - Config for berth

RATE - as reqd (VERN when near structure, -10 ft;  
below Z = -600)

BRAKES - OFF (tb-OFF)

MODE - ORB LD, ENTER

A6U

DAP: free drift

Mnvr to berthed position:

SPASI:

X	Y	Z	PITCH	YAW	ROLL	PL ID
-895.0	0.0	-413.7	360.0	360.0	360.0	1
SY	SP	EP	WP	WY	WR	
-29.1	69.0	-123.0	-45.8	16.9	-129.5	

**GRO:**

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1089.6	0.6	-414.1	0.3	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
-5.5	65	103.9	50.1	-10.5	-65	

**HST:**

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1010.4	-90.9	-414.1	360.0	359.5	0.0	1
SY	SP	EP	WP	WY	WR	
21.5	86.8	128.7	-55.5	18.1	137.8	

**IBSS:**

X	Y	Z	PITCH	YAW	ROLL	PL ID
-890.4	-94.7	-415.6	0.0	270.0	0.0	1
SY	SP	EP	WP	WY	WR	
-6.8	73.8	-117.7	-46.7	4.4	26.3	

**LDEF:**

X	Y	Z	PITCH	YAW	ROLL	PL ID
-892.2	-14.2	-399.6	0.4	359.7	90.0	1
SY	SP	EP	WP	WY	WR	
-0.1	79.2	-127.9	-41.2	-40.5	-160.4	

**SPARTANH:**

X	Y	Z	PITCH	YAW	ROLL	PL ID
-712.1	-0.5	-423.1	359.7	359.6	360.0	1
SY	SP	EP	WP	WY	WR	
-79.4	79.7	-137.8	-80.9	8.1	192.5	

A8U

BRAKES - ON (tb-ON)

4. PREP FOR UNGRAPPLE

To relieve strain:

DAP: VRCS or free drift

BRAKES - OFF (tb-OFF)

MODE - TEST, ENTER

Wait 5 sec, then

BRAKES - ON (tb-ON)

5. UNGRAPPLE

A7U CCTV, RMS/Wrist - zoom out

CRT PL ID - ITEM 3 +0 EXEC  
INIT ID - ITEM 24 +0 EXEC

DAP: VRCS or free drift

A8U RATE - VERN (RATE MIN tb-ON), when within  
10 ft of structure

BRAKES - OFF (tb-OFF)

MODE - END EFF, ENTER

CAUTION

Monitor EE tb timing to  
prevent EE motor burnout

If single joint,  
Perform Manual EE Release

EE MODE - AUTO

RELEASE sw - depress (mom)

When OPEN tb - gray,  
Mnvr arm clear of GF, orbiter, payload

X	Y	Z	PITCH	YAW	ROLL	PL ID
-908.2	13.7	-494.7	270.0	0.0	180.5	0

✓ RIGID CLOSE CAPTURE  
DERIGID OPEN EXTEND

CRITICAL TIMES (28 sec total):

DERIGID tb - gray, 5 sec max,  
then

OPEN tb - gray, 3 sec max,  
then

EXTEND tb - gray, 20 sec max

BRAKES - ON (tb-ON)

EE MODE - OFF

✓MODE - not DIRECT  
JOINT - CRIT TEMP

[ 6. MNVR TO PRE-CRADLE POSITION

A7U

CCTV - Config for Mnvr to Pre-Cradle

SM 94 PDRS CONTROL
--------------------

PL ID - ITEM 3 +0 EXEC  
INIT ID - ITEM 24 +0 EXEC

✓RATE - as reqd (VERN within 10 ft)  
✓BRAKES - OFF (tb-OFF)  
MODE - ORB UNL, ENTER

X	Y	Z	PITCH	YAW	ROLL	PL ID
-1261.2	-145.7	-551.4	4.7	1.7	359.9	0
SY	SP	EP	WP	WY	WR	

BRAKES - ON (tb-ON)  
MODE - not DIRECT  
JOINT - CRIT TEMP

**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix B: Part Task Design Document and User Manual**

---

<b>APPENDIX B: PART TASK DESIGN DOCUMENT AND USER MANUAL</b>
--



---

<b>TABLE OF CONTENTS</b>
--------------------------

Index of Tasks.....	2
System Functioning, Record Keeping, etc.....	4
Evaluation and Scoring .....	5
Ordering of Part Tasks during Tutorial Mode.....	10
Overview: Arm Operation Skills .....	16
Breakdown of Part Tasks for Arm Operation Skills .....	18
Part Tasks for Planning Skills .....	20
Part Tasks for Manipulation Skills .....	55
Part Tasks for Monitoring Skills .....	61
Part Tasks for Procedural Skills.....	78
Appendix A: Changing the Editable Parameters .....	A-1
Appendix B: A Listing of the Editable Parameters.....	B-1

## INDEX OF TASKS

*Task 1.2.1a: Exercise: Identifying Reference Points in 3 Dimensions.....	22
*Task 1.2.1b: Identifying Reference Points in 3 Dimensions.....	23
*Task 1.2.1c: Payload IDs.....	24
*Task 1.3.1.1a: Exercise: Translational Movement in ORB UNL Mode using BACS.....	27
Task 1.3.1.1b: Translational Movement in ORB UNL Mode using BACS.....	28
*Task 1.3.1.2a: Exercise: Rotational Movement in ORB UNL Mode using RACS.....	30
Task 1.3.1.2b: Rotational Movement in ORB UNL Mode using RACS.....	30
Task 1.3.1.3a: Integrating THC and RHC Movement in ORB UNL Mode using BACS and RACS.....	32
Task 1.3.1.4b: Translational Movement in ORB LD Mode using BACS.....	334
Task 1.3.1.5b: Rotational Movement in ORB LD Mode using RACS.....	35
Task 1.3.1.6a: Integrating THC and RHC Movement in ORB LD Mode using BACS and RACS.....	37
*Task 1.3.2.1a: Exercise: Translational Movement in END EFF Mode using ECS.....	39
Task 1.3.2.1b: Translational Movement in END EFF Mode using ECS.....	40
*Task 1.3.2.2a: Exercise: Rotational Movement in END EFF Mode using RACS.....	41
Task 1.3.2.2b: Rotational Movement in END EFF Mode using RACS.....	42
Task 1.3.2.3a: Integrating THC and RHC Movement in END EFF Mode.....	44
*Task 1.3.3.1a: Exercise: Translational Movement in Payload Mode using PCS.....	45
Task 1.3.3.1b: Translational Movement in Payload Mode using PCS.....	46
*Task 1.3.3.2a: Exercise: Rotational Movement in Payload Mode using PCS.....	48
Task 1.3.3.2b: Rotational Movement in Payload Mode using PCS.....	49
Task 1.3.3.3a: Integrating THC and RHC Movement in Payload Mode.....	50
*Task 1.4a: Visual Recognition of Arm Positions/Attitudes.....	53
*Task 2.2: Developing More Precise Arm Control using the THC.....	56
*Task 2.3: Developing More Precise Arm Control using the RHC.....	57
*Task 2.4: Developing More Precise Arm Control using Both Hand Controllers Simultaneously.....	58
Task 3.1a: Recognizing Singularities.....	62
Task 3.1b: Visualization of and Driving the Arm into Singularities.....	64
Task 3.2a: Recognizing Reach Limits.....	67
Task 3.2b: Visualization Of and Driving the Arm Into Reach Limits; Beginning Camera Skills.....	68
*Task 3.2c: Avoiding Reach Limits; Beginning Camera Skills.....	70
*Task 3.4a: Demonstration: Using Cameras for Specific Tasks.....	73
*Task 3.4b: Finding Optimal Camera Views.....	76
*Task 4.1a: Exercise: Use of SINGLE Mode; Developing Camera Skills.....	78
*Task 4.1b: Resolving Reach Limits; Developing Camera Skills.....	79



---

*Task 4.2a: Driving Into and Resolving Software Stops.....	81
*Task 4.2b: Avoiding Software Stops; Beginning Camera Skills.....	82
Task 4.3a: Grapple Procedure; Camera Skills .....	84
Task 4.4a: Ungrapple Procedure; Camera Skills .....	86
Task 4.5a: Berthing a Payload; Camera Skills .....	88
Task 4.6a: Unberthing a Payload; Camera Skills .....	91
Task 4.7a: Unloaded Arm Phasing; Camera Skills.....	93
Task 4.7b: Loaded Arm Phasing; Camera Skills.....	95
Task 4.8a: Whole Task: Deployment.....	96
Task 4.9a: Whole Task: Retrieval.....	98
*Task 4.10a: Uncradle .....	99
*Task 4.10b: Cradle .....	100

\* Tasks marked with asterisks will NOT be implemented in this phase of the project due to lack of project resources.

The P2T2 Intelligent Trainer is expected to be used as a research instrument (effectiveness of ITSSs), as a diagnostic screening device, and as a basic skills trainer. In order to support instructors, students, and researchers, the following capabilities are available:

1. **Modes of operation:** The P2T2 Intelligent Trainer can function in three different ways.
  - **Simulation mode:** This is the normal "P2T2" simulator without any kind of instructional capability or evaluation/scoring. No records are kept of student performance. Simulation mode is a free-play environment.
  - **Tutorial mode:** Tutorial mode represents an ordered sequence of tasks. All new students must master all part tasks in the prescribed sequence. The order of the part tasks is given in the section of this document titled "Ordering of Part Tasks during Tutorial Mode."
  - **Skill Maintenance mode:** After a student has mastered all part tasks, he/she is automatically placed in skill-maintenance mode. The state of the student's knowledge per part task (as recorded in the Student Model) is changed from "mastered" to "unknown." The system then presents the student with a whole task and tries to determine any weaknesses in the student's knowledge or skills. Weaknesses are then addressed using the relevant part task at a median level of difficulty.
2. **Timer:** The timer can be turned off at a global level (as an editable parameter; see Appendices A and B). Time measurements are still kept in the student database and, unless the efficiency measurement is changed per part task (see #2 below), time will still count as part of the efficiency measure. However, students will not see a timer working.
3. **Time/Efficiency:** The percentage that time (the amount of time a student takes to complete one trial) contributes to the efficiency score per task can be modified per part task. (If the current efficiency measurement for a part task is solely related to time and the percentage of time is changed to zero, then efficiency will not be measured for that task.)
4. **Editable Parameters:** Proficiency values for many different measurements and components of measurements can be changed per part task. For example, instead of three consecutive error-free performances, the instructor to change the value to five. Number of trials that must be mastered per LOC, time limits, and visual-aid buttons (as well as length of time the visual aid returns) are some of the quantities that are editable.

See Appendix A for an overview of editable parameters and detailed instructions on changing them. See Appendix B for a listing of all editable parameters.

5. Database: Simple database files containing student evaluative data per task number were created. There are actually two files per student per part task:

/data/students/<LASTNAME FIRSTNAME MI>/<taskname>AllScores.dat

This file contains appended data for each time the student has taken a particular part task. For example, <taskname> might be GrappleAllScores.dat.

/data/students/<LASTNAME FIRSTNAME MI>/<taskname>OverallScores.dat

t This file contains the best and average scores for the student per LOC per part task.

These files are simple repositories of data. They do not work as relational databases. Eventually, for large training applications, a formal database such as Informix should be used to perform statistical analyses, etc.

## Evaluation and Scoring

### OVERVIEW

This section tells how evaluation and scoring are done by the P2T2 Intelligent Trainer.

### EDITABLE PARAMETERS

Note that all mastery and advancement criteria, as well as a large number of other aspects of the P2T2 Intelligent Trainer, are changeable by instructors. These changeable values are called "Editable Parameters." See Appendices A and B for information on what can be changed and how to perform such tasks.

### MASTERY TRAINER

The P2T2 Intelligent Trainer is a *mastery* trainer. The system is made up of an ordered sequence of part tasks. Each part task contains a number of levels of complexity (usually 3-4, defined in this document). In turn, each level of complexity contains a number of trials (usually 3-5, but editable). Mastery is determined per trial and per part task, as well as per the entire set of part tasks.

**Per trial:** For each trial, the student must succeed according to the evaluation criteria as stated in this document or as changed in the Editable Parameters files. Default here is that if, for example, a student passes three of five trials and then fails a trial, s/he must start that level of complexity over (unless this criteria is changed in the Global Parameters file).

**Per part task:** For each part task, students must succeed according to the task's advancement criteria in order to proceed to the next part task. Advancement criteria are stated in this document per part task, although they may be changed in the Editable Parameters files.

**Proficiency advancement:** If a student passes the first two trials of any level of complexity (LOC) without preceding or intervening failure, he or she can "proficiency" that LOC and proceed to the next LOC of the task, or to the next task. (The number of trials required to proficiency an LOC is editable at the global level.)

A student must master every part task in order to enter the skill maintenance mode of trainer operation (see the previous section for information on trainer modes).

### PERFORMANCE INDICATOR

A performance indicator is used in most part tasks except those where advancement to the next task is student-controlled (i.e., exercises). The purpose of the performance indicator is to let the student know at all times where he/she is in relation to completing the trial (in terms of time remaining, when applicable), in relation to completing the level of complexity, and of completing the part task itself.

Number of trials per complexity level and number of complexity levels shown on the performance indicator vary depending on the part task. Also, the speed at which the timer moves varies per trial/task (i.e., if a trial must be performed in 60 seconds, the timer will completely cover the earth under it in 60 seconds; if the trial is allowed 3 minutes, the timer will cover the earth in 3 minutes; etc.). These measures are defined per task in this document and changeable per editable parameters.

## PERFORMANCE EVALUATION MEASURES

Measures for performance evaluation were developed with the help of Dr. Debra Johnson, University of Houston. In addition, at the advice of our consultant, Dr. J. Wesley Regan of Brooks Air Force Base, HRL, time limits were added to most tasks as a component of efficiency, even where they are not normally critical. The rationale for this is that people who can perform the tasks quickly generally have greater ability (aptitude) for the task: these people are more likely to succeed when time is an evaluative measure. Time can be turned off completely at the global level, or its contribution to the efficiency measure can be increased or decreased per part task.

Generally, all performance evaluation measures are editable parameters. See Appendices A and B.

## SCORING

Scoring is performed for up to five measures (where applicable) per part task: accuracy, efficiency, safety, procedural correctness, and camera usage. Almost all aspects of scoring are tunable in the editable parameters files.

At the end of each trial, the student is given a detailed description of his/her performance, including the individual scores for each measure as well as procedural steps that were omitted, etc.

**Overall score:** An overall score is determined per trial for each part task based on task-specific parameters. For example, accuracy can count for 25% of the overall score, efficiency for another 25% of the overall score, etc. Once the final score is determined, a global parameter determines whether the student has passed the trial. For example, the global parameter may require a student to attain a score of 75% or greater to pass the trial.

**Accuracy:** Accuracy is a function of how close the student is to the goal position, usually determined at the point the student presses the DONE button. Translation and rotation, when both are applicable, always count for 50% of the accuracy score (not editable).

**Translation:** A translational "allowance" (e.g., 5 inches) is defined in the global parameters file. If the student is at the exact goal position, the score is 100%. If the student is right at the allowance (e.g., is off by a total of 5 inches, but not more), his/her accuracy score is equal to the passing score (which is a global editable parameter, currently 75%). Other scores are derived using this scale.

**Rotation:** Pitch and Yaw count as one measure, and roll as a second measure in the rotational component of accuracy. Pitch and yaw accuracy are determined by the angle between the goal state and the final position of the arm; roll is the same. As with translation, a global parameter defines a rotational allowance for student performance.

**Efficiency:** Efficiency is a function of path traveled by the arm and, when used, time. The amount that each measure contributes to the total efficiency score is editable for each part task.

**Path:** Each trial of a part task has a minimum path the arm could travel in order to obtain the goal position. Also, an editable parameter is used as a scaler; for example, the scaler could be 1.5. (This scaler is editable for each relevant part task.) If the student's distance traveled by the arm equals the minimum path, the path portion of the efficiency score is 100%. However, if the student's path equals the minimum path multiplied by the scaler, then the path efficiency score is 75%. (A scaler of 1.5 means the student could move the arm an additional 50% of the minimum possible path and still pass the path portion of efficiency.) Other scores are derived using this scale.

**Time:** Editable parameters for each part task determine the value of time in creating an efficiency score. In addition, the amount of time allowed for acceptable performance of each trial is editable for each task. If the student uses exactly the amount of time allowed for the trial, his/her score is whatever the passing score is. (Passing score is the minimum score a student can get to pass a portion of a task or the entire task. It is a global parameter.) If the student uses one-half the amount of time allowed for the trial, his/her score is 100%. Other scores are derived using this scale.

**Safety:** A student starts with a safety score of 100. For every safety violation, a number determined by a task-specific editable parameter is deducted from the score. For example, the student may hit a singularity and have 25 points deducted from his/her score for that trial, leaving a safety score of 75.

**Procedural correctness:** Like safety, a student starts with a score of 100. For every omitted, misplaced, or wrong procedural step, a number determined by a task-specific editable parameter is deducted from the score. Note that expected procedures are defined in a procedural checklist document especially created for the P2T2 Intelligent Trainer.

**Camera usage:** For each part task where camera usage is monitored, the percentage of time the student must have a particular view of the arm (or related structures) per trial is determined by an editable parameter. If the student has the correct view 100% of the entire time, his/her score is 100. If the student has the correct view for the required (editable) amount of time, the score is 75 (or the current value of the passing score, a global parameter).

## STUDENT EVALUATION GRAPH

After every trial, a student can access an evaluation graph. This graph shows scores both graphically and numerically for each evaluative measure—i.e., safety, efficiency, accuracy, procedural correctness, and camera usage. In addition, the best, average, and current scores are shown. "Best" and "Current" refer to the current LOC; "Average" refers to the entire part task, all LOCs. From this screen, the student can also press a button to review the textual performance evaluation received at the end of the last trial.

## Ordering of Part Tasks

### OVERVIEW

Working with our consultant, Dr. J. Wesley Regian, we selected a theory of instructional design called Elaboration Theory as the basis of ordering part tasks in Tutorial Mode. Elaboration Theory, developed by Charles M. Reigeluth and M.D. Merrill\*, is concerned with the *macro* level of instructional design (i.e., the selection, sequencing, synthesizing, and review of training material).

In this section, we present a brief overview of Elaboration Theory, our rationale for a particular ordering of the RMS part tasks according to the tenets of Elaboration Theory, and a list of all the tasks in the order of presentation to the student during Tutorial Mode.

Note that the order for presentation to the student is different from the ordering of part tasks in this document. This document divides the tasks into categories of Planning, Arm Manipulation, Monitoring, and Procedural Skills. The Task number associated with the task names throughout this document (e.g., Task 3.2.2.a) denotes their category and order in this document. The TM number associated with the task names below (e.g., TM5) denotes their presentation order in Tutorial Mode.

### ELABORATION THEORY

The instructional design guidelines of the Elaboration Theory are based both on an analysis of the structure of knowledge and on an understanding of cognitive processes and learning theories. Many different learning and instructional design theories were incorporated into the Elaboration Theory by Reigeluth and Merrill. The authors claim that training designed using Elaboration Theory results in enhanced achievement, retention, transfer, and motivation.

The best way to exemplify the Elaboration Theory might be to give an analogy of how instruction would function using this approach. The analogy is that of looking at a picture through a zoom lens. (This analogy is taken from the NSPI Journal article.) "The instruction begins with the *wide-angle view*, which shows only the major parts of the picture and their interrelationships. There is little detail. You can gradually increase the detail and complexity of the parts which particularly interest you by *zooming in* on them....After zooming in one level on one part of the picture, you must zoom back out to integrate that information with the larger whole and to review previous instruction. Then you can go back and zoom in to a second level on the same part, this time seeing even greater complexity and detail. Or instead you could zoom in to the first level on a different part."

---

\* Reigeluth, C.M., & Stein, F.S. The Elaboration Theory of Instruction. In C.M. Reigeluth (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status*. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1983.

Reigeluth, C.M., & Rodgers, C.A. The Elaboration Theory of Instruction: Prescriptions for Task Analysis and Design. *NSPI Journal*, February, 1980, 16-25.



According to Elaboration Theory, there are three fundamental types of content: concepts, procedures, and principles. This distinction is extended to three major types of relationships in subject matter: conceptual, theoretical, and procedural. The theory prescribes that the elaborative sequencing of the content (the zooming-in process) should be based on just one of these three types of content relationships. In the case of RMS training, procedural relationships seem the most applicable. Procedural relationships are interrelated chains of procedures, and all of the RMS part tasks rely on the student performing RMS procedures.

In designing a training sequence using Elaboration Theory, one first selects an *epitome* of the training content. An epitome involves teaching a small number of (in this case) procedures at the application level: i.e., have the student perform an actual task that is typical of the domain and that is at a low level of complexity. Other design guidelines include:

- . Allow for early application of the procedures to realistic cases
- . Gradually add more complexity to the lessons (i.e., part tasks)
- . Require the student to perform systematic review by periodically integrating the most recent learning with earlier learning
- . Require mastery of one level of complexity before allowing the student to continue to greater complexity

Note that the part tasks were developed using the theory of mastery learning: the student must master each level of complexity in every part task before proceeding to the next level of complexity or the next part task. Note also that these levels of complexity further reflect the simple-to-complex ordering sequence advocated by the Elaboration Theory.

## ELABORATION THEORY AND RMS TRAINING

In applying the Elaboration Theory to RMS training, we determined that the basic epitome of RMS training is flying the arm. All other tasks and procedures, with the possible exception of coordinate systems (designated as epitome #2) and camera skills (epitome #3), are directly related to this single task. Even grappling/ungrappling and berthing/unberthing could be viewed as highly-specialized, highly-skilled aspects of flying the arm.

For reasons expressed below, the RMS part tasks were ordered in the following manner:

1.     **Epitome #1 task:** Have the student perform a simple fly-to  
  
A very short fly-to using ORB UNL mode was selected as the epitome of RMS operations, since ORB UNL mode is the most intuitive mode in terms of how the part tasks work\*. The supporting tasks have the student fly the arm to very short target locations indicated with a ghost arm using first the THC, then the RHC. (TM 1-4)
2.     **Integrating task:** The student uses both hand controllers simultaneously in ORB UNL mode for short fly-to tasks (TM5)
3.     **Add complexity:** Student performs short fly-to tasks in the other modes and coordinate systems using first one hand controller, then the other. (TM6-17)
4.     **Integrating task:** Another task integrates the use of hand controllers simultaneously for END EFF, a mode very different than ORB UNL (TM18)
5.     **Add complexity, review previous learning:** Student performs a series of tracing tasks using first one hand controller, then the other; END EFF mode is used (TM19-20)
6.     **Integrating task:** Student performs a series of tracing tasks using both hand controllers simultaneously and END EFF mode (TM21)
7.     **Epitome #2:** Student learns to associate coordinates in BACS with specific points in the payload bay (TM22-23)
8.     **Add complexity, review previous learning:** Student learns that Payload IDs affect the point of resolution (POR) (TM24)
9.     **Integrating task:** Student learns to relate knowledge of coordinate systems and POR information to positions and attitudes of the arm (TM25)
10.    **Epitome #3:** Student learns optimal views of the RMS tasks using cameras, and begins to develop skills to select such views (TM26-27)
11.    **Integrating task, add complexity:** Student learns the grapple/ungrapple procedures and berthing/unberthing, which require more precise control of the arm and adjusting the cameras (TM28-31)

---

\* END EFF mode, when viewing out the wrist camera, is actually the most intuitive method of flying the arm. However, because the operator must superimpose the arm over a ghost arm image when performing the fly-to tasks, he/she cannot use the wrist camera in END EFF or the ghost arm target would not be visible.

12. Add complexity, review previous learning, integrate skills: Student learns to recognize singularities and reach limits; student performs tasks to deliberately encounter these problems; student performs tasks to avoid encountering these problems; continuing development of camera skills in relation to performing the specific tasks (TM32-36)
13. Add complexity: Student learns a new mode of operating the arm (Single mode) (TM37)
14. Integrate skills, add complexity: Student learns to use Single mode to resolve reach limits and software stops; continuing development of camera skills (TM38-40)
15. Integrate skills: Student performs long fly-to's that require using knowledge of coordinate systems and input modes, and avoiding reach limits and singularities; camera usage is monitored (TM41-43)
16. Integrate skills, review previous learning: Student must perform the entire deploy task and the entire retrieval task. These tasks integrate all three epitomes and all previously-learned skills and knowledge. (TM44-45)

#### **ORDER OF PART-TASK PRESENTATION TO THE STUDENT**

Based on Elaboration Theory, the part tasks described in this document will be presented to the student during Tutorial Mode in the following order:

- |            |  |
|------------|--|
| <b>TM1</b> | <b>Task 1.3.1.1a: Exercise: Translational Movement in ORB UNL Mode using BACS</b>          |
| <b>TM2</b> | <b>Task 1.3.1.1b: Translational Movement in ORB UNL Mode using BACS</b>                    |
| <b>TM3</b> | <b>Task 1.3.1.2a: Exercise: Rotational Movement in ORB UNL Mode using RACS</b>             |
| <b>TM4</b> | <b>Task 1.3.1.2b: Rotational Movement in ORB UNL Mode using RACS</b>                       |
| <b>TM5</b> | <b>Task 1.3.1.3a: Integrating THC and RHC Movement in ORB UNL Mode using BACS and RACS</b> |
| <b>TM6</b> | <b>Task 1.3.1.4b: Translational Movement in ORB LD Mode using RACS</b>                     |
| <b>TM7</b> | <b>Task 1.3.1.5b: Rotational Movement in ORB LD Mode using BACS</b>                        |
| <b>TM8</b> | <b>Task 1.3.1.6a: Integrating THC and RHC Movement in ORB LD Mode using BACS and RACS</b>  |
| <b>TM9</b> | <b>Task 1.3.3.1a: Exercise: Translational Movement in Payload Mode using PCS</b>           |

TM10	Task 1.3.3.1b: Translational Movement in Payload Mode using PCS
TM11	Task 1.3.3.2a: Exercise: Rotational Movement in Payload Mode using PCS
TM12	Task 1.3.3.2b: Rotational Movement in Payload Mode using PCS
TM13	Task 1.3.3.3a: Integrating THC and RHC Movement in Payload Mode
TM14	Task 1.3.2.1a: Exercise: Translational Movement in END EFF Mode using ECS
TM15	Task 1.3.2.1b: Translational Movement in END EFF Mode using ECS
TM16	Task 1.3.2.2a: Exercise: Rotational Movement in END EFF Mode using RACS
TM17	Task 1.3.2.2b: Rotational Movement in END EFF Mode using RACS
TM18	Task 1.3.2.3a: Integrating THC and RHC Movement in END EFF Mode
TM19	Task 2.2: Developing More Precise Arm Control using the THC
TM20	Task 2.3: Developing More Precise Arm Control using the RHC
TM21	Task 2.4: Developing More Precise Arm Control using Both Hand Controllers Simultaneously
TM22	Task 1.2.1a: Exercise: Identifying Reference Points in 3 Dimensions
TM23	Task 1.2.1b: Identifying Reference Points in 3 Dimensions
TM24	Task 1.2.1c: Payload IDs
TM25	Task 1.4a: Visual Recognition of Arm Positions/Attitudes
TM26	Task 3.4a: Demonstration: Using Cameras for Specific Tasks
TM27	Task 3.4b: Finding Optimal Camera Views
TM28	Task 4.3a: Grapple Procedure; Camera Skills
TM29	Task 4.4a: Ungrapple Procedure; Camera Skills
TM30	Task 4.5a: Berthing a Payload; Camera Skills
TM31	Task 4.6a: Unberthing a Payload; Camera Skills
TM32	Task 3.1a: Recognizing Singularities

**Ordering of Part Tasks during Tutorial Mode**

---

TM33	Task 3.1b: Visualization of and Driving the Arm into Singularities
TM34	Task 3.2a: Recognizing Reach Limits
TM35	Task 3.2b: Visualization Of and Driving the Arm Into Reach Limits; Beginning Camera Skills
TM36	Task 3.2c: Avoiding Reach Limits; Beginning Camera Skills
TM37	Task 4.1a: Exercise: Use of SINGLE Mode; Developing Camera Skills
TM38	Task 4.1b: Resolving Reach Limits; Developing Camera Skills
TM40	Task 4.2a: Driving Into and Resolving Software Stops
TM41	Task 4.2b: Avoiding Software Stops; Beginning Camera Skills
TM42	Task 4.7a: Unloaded Arm Phasing; Camera Skills
TM43	Task 4.7b: Loaded Arm Phasing; Camera Skills
TM44	Task 4.8a: Whole Task: Deployment
TM45	Task 4.9a: Whole Task: Retrieval

### Overview: Arm Operation Skills

Each time the arm is manipulated, the arm operator employs a process to ensure safety, efficiency, and accuracy. This process is cyclical in nature and involves three stages: planning, arm manipulation, and monitoring.

#### PLANNING

The planning stage of arm manipulation requires the use of primarily cognitive skills. In performance, some astronauts do more planning than others: most experienced arm operators do some planning prior to moving the arm, but observation of novice arm operators, as well as one of our three astronauts observed during knowledge acquisition, shows that some do not plan sufficiently if at all. Minimally, the arm operator creates a mental image of the goal position, attitude and arm configuration. He/she then imagines how to move the arm from its current configuration into the goal state, presumably taking safety and efficiency factors into consideration.

The following steps comprise the planning stage of arm operation. The skills associated with each step should be part of the training.

1. Create a visual image of the goal configuration
2. Consider the movements necessary to place the arm in the target configuration to achieve the desired POR
3. Consider the kinds of problems that might occur during these arm movements (singularities, reach-limits, and safety violations)
4. Consider alternative movements that would reduce the possibility of encountering such problems
5. Develop a strategy (applicable modes of operation, series of inputs) to place the arm efficiently and safely into the target location and configuration

#### MANIPULATION

The manipulation process is primarily psychomotor in nature. The arm operator must possess the hand-eye coordination necessary to physically control the movement of the arm. (He/she must also understand how the two hand controllers work: which movement of each controller causes what movement of the arm. These skills were covered above under "Planning.") Manipulation involves understanding and correct application of:

- . inputs to the THC
- . inputs to the RHC
- . inputs to the THC and RHC simultaneously

## **MONITORING**

The monitoring process is visual information processing. To ensure safety, the operator must be aware of the position of the arm structure in relation to the Orbiter and other objects such as the payload. When monitoring, the arm operator gets feedback to assess the effects of and adjust hand-controller inputs in order to achieve the target arm configuration. Monitoring also decreases the possibility of encountering singularities and reach-limits while configuring the arm. Monitoring involves:

1. Observe the movement and relative location of the arm at all times
2. Ensure that arm structures do not threaten the Orbiter or other structures
3. Avoid singularities and reach limits

## **PLAN REVISION (if necessary)**

The manipulation, monitoring, and plan revision cycle repeats until the arm operator has placed the EE or payload in the goal position and attitude.

- . Based on current input, revise the plan when necessary

### Breakdown of Part Tasks for Arm Operation Skills

The skills necessary to perform payload deployment and retrieval tasks can be divided into the following four categories:

1. Planning skills
2. Manipulation skills
3. Monitoring skills
4. Procedural skills

In this section, we further decompose these skills and relate them to specific aspects of RMS operation. We also present certain over-all explanations. In the next section, tasks for teaching these skills are described.

#### 1. PLANNING SKILLS

- 1.1 Associated knowledge of coordinate systems
  - 1.1.1 Orbiter Body Axis Coordinate System
  - 1.1.2 End Effector Coordinate System
  - 1.1.3 Orbiter Rotation Axis Coordinate System
  - 1.1.4 Payload Coordinate System
- 1.2 Spatial skills (identification of points in space related to the Orbiter)
  - 1.2.1 2-dimensional space
  - 1.2.2 3-dimensional space
- 1.3 Visualization skills (creation of mental images)
- 1.4 Mode/input/response relationships (arm reaction to inputs in various modes)
  - 1.4.1 Orbiter Unloaded and Orbiter Loaded modes
    - 1.4.1.0 Using BACS
    - 1.4.1.1 THC input/response
    - 1.4.1.2 RHC input/response
  - 1.4.2 End Effector mode
    - 1.4.2.0 Using EE Coordinate System
    - 1.4.2.1 End Effector Coordinate System
    - 1.4.2.2 THC input/response
    - 1.4.2.3 RHC input/response



1.4.3 Payload mode

- 1.4.3.0 Using the Payload Coordinate System
- 1.4.3.1 THC input/response
- 1.4.3.2 RHC input/response

**2. MANIPULATION SKILLS**

- 2.1 THC control
- 2.2 RHC control
- 2.3 Simultaneous control

**3. MONITORING SKILLS**

- 3.1 Singularity avoidance
- 3.2 Reach-limit avoidance
- 3.3 Resolving software stops
- 3.4 Camera skills
- 3.5 Using Spec 96

**4. PROCEDURAL SKILLS**

- 4.1 Mode selection
- 4.2 Singularity recovery
- 4.3 Reach-limit recovery
- 4.4 Software-stop recovery
- 4.5 Grappling
- 4.6 Ungrappling
- 4.7 Berthing
- 4.8 Unberthing
- 4.9 Fly-to-positions and attitudes
- 4.10 Deployment
- 4.11 Retrieval

## Part Tasks for Planning Skills

### 1. PLANNING SKILLS

Planning involves the advance development of optimal arm manipulation strategies. The operator must understand the goal arm state in terms of both position and attitude, the trajectory of the arm (and parts of the arm) while moving to the goal state given the initial position of the arm, how the arm operates in the various modes, and the arm configurations encountered while moving to the goal state that might result in problems (i.e., singularities, reach limits, or safety violations). A well-thought-out plan developed *prior* to beginning a task can contribute to a more efficient and safe performance. (Note that singularities, reach limits, etc., are taught in the Monitoring section that follows.)

#### Advance Organizer

#### PLANNING SKILLS

"Purpose: Successful, efficient, and safe operation of the RMS depends on the development of planning skills. Developing a strategy to complete each task the P2T2 Intelligent Trainer presents will improve your performance.

Task: In the following tasks, you will demonstrate your knowledge and understanding of various RMS concepts. Mastery of these concepts provides a foundation for the RMS skills you will develop in other tasks and on other simulators.

Evaluation criteria: Evaluation criteria are described at the beginning of each task. Most tasks are evaluated for safety, efficiency, and accuracy, as well as procedural correctness. Each task is composed of several levels of increasing complexity or difficulty, which in turn are composed of three to five "trials." After mastering each level of complexity, you can continue to the next task.

Hint: Understanding the following knowledge areas and using some basic planning tactics prior to moving the arm will help you succeed in performing these tasks:

- . Understand the different coordinate systems.
- . Know how the arm responds to various control modes.
- . Visualize the goal arm state in terms of both position and attitude.
- . Determine the trajectory of each segment of the arm while moving to the goal state from the initial arm configuration.
- . Envision the arm configurations encountered while moving to the goal state that may result in problems such as singularities, reach limits, or safety violations. "

## 1.1 COORDINATE SYSTEMS (KNOWLEDGE)

Since the arm operates relative to various coordinate systems, the operator must know these coordinate systems in order to create a realistic plan of operation. Prior to use of the P2T2 Intelligent Trainer, the student has studied definitions, diagrams, and explanations of the various Orbiter/arm coordinate systems. We therefore assume the student has a basic understanding of the location of the origin and the relationships between the axes in the various coordinate systems prior to work on the P2T2.

- 1.2 Orbiter Body Axis Coordinate System
- 1.3 End Effector Coordinate System
- 1.4 Orbiter Rotation Axis Coordinate System
- 1.5 Payload Coordinate System

## 1.2 ORBITER BODY AXIS COORDINATE SYSTEM (SPATIAL SKILLS)

To create a plan that contributes to efficient arm manipulation, the operator must be able to locate points in space relative to the Orbiter and the arm. This requires the creation of a mental image of a 3-dimensional coordinate system and the visualization of where a particular set of coordinates is located within that coordinate system (in relation to the Orbiter). Such visualization always takes place in the Orbiter Body Axis Coordinate System (BACS). This visualization provides the student with a target destination for the POR relative to the current position and attitude, and the student can then plan EE or payload movements accordingly.

### • Advance Organizer

### • ORBITER BODY AXIS COORDINATE SYSTEM (BACS) SPATIAL SKILLS

"Purpose: Identify the spatial relationships of the RMS point of resolution (POR). One of the basic skills for planning efficient arm manipulation is the ability to identify various locations in the orbiter payload bay. A mental image of the coordinate system's origin and other reference points can aid in the positioning of the POR.

Task: Several tasks follow to increase your knowledge of the orbiter body axis coordinate system.

Evaluation criteria: In these tasks, you are simply told whether your answers or solutions are right or wrong.

Hint: When identifying points in 3-dimensional space, make note of nearby reference points that have known locations (for example, the MPMs). Offset the POR by the difference and compare distances to objects of known size nearby."

### 1.2.1 3-DIMENSIONAL SPACE

#### Task 1.2.1a. Exercise: Identifying Reference Points in 3 Dimensions

In this task as well as 1.2.1b, a "perfect" side view of the Orbiter in gods-eye perspective is used as well as a gods-eye view from directly overhead; in other words, these views are somewhat 2-dimensional.

A small object is shown on the views of the Orbiter (same object and location, different perspectives); however, the arm is not visible.

A window containing the letters "X," "Y," and "Z" is shown with a number to the right of each letter representing the true values for each coordinate relative to the object. For example:

X: -550  
Y: +250  
Z: -310

The student uses the THC to move the object around, and can watch the XYZ values change.

The student plays with this exercise as long as desired. The screen is prominently labeled with the mode of operation and the coordinate system.

Advance Organizer:

"Purpose: To develop a repertoire of known points in the orbiter payload bay and associated coordinates.

"Task: Use the THC to position the point of resolution (POR) at various locations in the payload bay.

Evaluation criteria: This task is not evaluated.

Hint: Select points such as the center of the payload bay, tip of the tail, top of the bulkheads, MPMS, etc. as reference points."

Levels of Complexity: NA

Evaluation: NA

Advancement criteria: Determined by the student: A "NEXT TASK" button allows the student to end the exercise and begin the actual part task.

### Task 1.2.1b: Identifying Reference Points in 3 Dimensions

As in task 1.2.1a, a 3-dimensional view of the Orbiter is used. Specific "landmark" points on the Orbiter are indicated on the screen one at a time (e.g., the center of the payload bay, top of the longeron, top of the bulkheads, above the MPM pedestals; see diagram in PDRS training manual). In addition, a window containing the letters "X," "Y," and "Z" is presented with three numbers to the right of each letter representing three X coordinates, three Y coordinates, and three Z coordinates. For example:

X:	+720	+300	-720
Y:	+250	-450	-250
Z:	-310	+520	+450

One number in each group of three is the correct coordinate for the point depicted on the screen. The screen is prominently labeled with the mode of operation and the coordinate system.

As each reference point is placed on the screen, the student must indicate the X, Y, and Z coordinates that identify that point by using the mouse to move a cursor over the correct coordinates (one at a time) and clicking the mouse button to confirm each number selected.

For each correct coordinate selection, the number changes color to green (or is highlighted with green). After all three numbers are selected correctly, a new point appears. If a student selects an incorrect coordinate, the word "No" appears briefly. By the fourth LOC, the student must type in the coordinate values.

#### Advance Organizer:

"Purpose: Test your knowledge of the BACS coordinates for various points in the payload bay.

Task: Identify BACS coordinates of a POR. You will choose from three choices for each X, Y, and Z. Use the mouse for your selection. (In later levels of difficulty, you will type in the values yourself using the keyboard.)

Evaluation criteria: Number of consecutive correct selections. When all three numbers for X, Y, and Z are correct, a new POR will be displayed for testing.

Hint: If you make a mistake, complete the task. You will get another attempt to make the right choice."

Levels of Complexity:

- (1) Use landmark locations (as described above) and distractor values that are obviously wrong (e.g., greater than 50% of each other)
- (2) Use non-landmark locations and distractor values 25%-45% of each other
- (3) Use non-landmark locations and distractor values 15-20% of each other
- (4) Have student type in the values for XYZ

Evaluation: Number of trials required to attain error-free performance. Selection must be made in 15 seconds or less.

Advancement criteria: Student correctly identifies five consecutive points per LOC without any errors and within the time constraints.

**Task 1.2.1c Payload IDs**

Since the student will have panel A8 available on most of the part tasks, some confusion may result when reading coordinates in PL ID 0 ("payload identification zero," no payload) and PL ID 1, since the point of resolution is different.

The first part of this task is a demonstration. For each image, the POR is highlighted by a small red ball, and each window is labeled with mode and payload ID. Panel A8 is also displayed for each arm. These examples are basically designed in pairs: for each arm *without* a payload, a corresponding screen must show the same arm *with* a payload. The student simply "thumbs through" several examples of arms in different positions with and without payloads using a NEXT EXAMPLE button; but until the NEXT TASK button is used, the examples will simply restart at the beginning (i.e., after example #6, student will see example #1 again).

Once the student selects the NEXT TASK button, the student gets related exercises. In this task, separate screens must be used for each example, since only one arm (i.e., either with a payload or without) can be displayed at a time per screen (P2T2 restriction). When the timer is used, the student gets 30 seconds to view the first example (or can press the COMPARE button sooner), and then must select an answer within 30 seconds of seeing the second example.

A window is superimposed over panel A8 labeled with the coordinate system, mode, and payload ID (e.g., PAYLOAD ID 0) and, below that, Target Coordinates, and shows coordinates of the correct answer. Two answer buttons in this window are labeled "This arm" and "Other arm." (Some delay is expected when loading the alternative arm; thus, we are trying to avoid *forcing* the student to toggle back again when selecting.) On one screen, the student sees an arm in two views (one from the aft window) with a large payload attached; on the other

screen the arm has no payload. The two arms are otherwise in the exact same position and attitude. The fourth window is used for the performance indicator and feedback. The student must simply use the mouse to "press a button" to select the arm that corresponds to the target coordinates; this selection must be made before the timer runs out (when the timer is used), or the student is judged incorrect. Indicate whether the student was correct or not per trial. Allow student to toggle back to the first example or to select "NEXT TASK" for the next trial or LOC.

Advance Organizer:

"The purpose of this task is to help you become aware of the difference in coordinates between Payload Identification 0 (PL ID 0) and PL ID 1. PL ID 0 is an arm with NO attached payload, whereas PL ID 1 has an attached payload. If you recall that the POR is defined as the tip of the EE when there is no payload, and as a point in space--usually a point within the payload--when a payload is attached to the arm, you will understand the meaning of the associated coordinates.

"This task has two sections: first, a demonstration, and second, an exercise. In the demonstration, you will see related pairs of arms and panel A8 readouts. Study the difference in coordinates between PL ID 0 and PL ID 1 for these examples using the NEXT EXAMPLE button. When ready for the exercise, press the NEXT TASK button.

"In the exercise, you will use two screens. One screen shows the arm without a payload, and one shows the arm with an attached payload. Target coordinates are given. Simply select which of the two arms exemplifies the target coordinates.

"Also in the exercise, you are evaluated for correctness. However, a time limit may be applicable: if the timer is running, you have 30 seconds to view the first arm, and 30 seconds after the second arm appears to make a selection."

Levels of Complexity:

- (1) As above
- (2) Use a midsized payload
- (3) Remove any visual aids

Evaluation: Accuracy: Correct/incorrect; Efficiency: student gets 30 seconds to view first arm, then must select an answer 30 seconds after second screen has plotted

Advancement criteria: Three consecutive correct trials per Level of Complexity (LOC) without intervening error and within time constraints.

### 1.3 MODE/INPUT/RESPONSE RELATIONSHIPS

The key to skills needed for controlling the arm in the various modes includes: 1) knowing under what conditions each is most appropriate, 2) knowing which PL ID is applicable along with the corresponding POR, and 3) knowing the movements that result from the various hand-controller inputs when in each mode.

Illustrations in the PDRS workbooks show the modes of operation and related coordinate systems as well as the THC/RHC input responses. These illustrations depict the arm from a perspective that is essentially a "gods-eye" view, because this perspective is conducive to visualizing the various relationships. In these exercises and tasks, the student first learns the relationships between controller inputs and arm responses in each mode separately. Then, a deeper understanding is achieved by teaching the student to distinguish these relationships from one mode to another.

#### Advance Organizer

#### INPUT MODES AND PLANNING

"One of the skills needed for good RMS task planning is understanding how the arm functions in the different modes of operation. This includes knowing: 1) the conditions in which each mode is appropriate, 2) the applicable payload ID (PL ID), as well as the location of the corresponding POR, and 3) the movements that result from the hand-controller inputs when in each mode.

"In the following series of exercises and tasks, you will learn the relationships between controller inputs and arm responses for each hand controller per mode separately; for example, you will practice using the THC only in ORB LD mode. Then, both hand-controllers are used simultaneously to perform simple fly-to exercises. Later, you will put all this knowledge and skill together to perform larger and more complex RMS tasks."

#### 1.3.1 ORBITER UNLOADED MODE (ORB UNL) AND ORBITER LOADED MODE (ORB LD)

Orbiter Unloaded mode is used primarily to move the arm when no payload is attached, while the Orbiter Loaded mode is used primarily for berthing and unberthing operations. The major difference between these two modes is the payload ID. (PL ID 0 is used for ORB UNL mode, which places the POR in the tip of the EE, and PL ID 1 is used for Orbiter Loaded, which usually defines the POR as a point within the payload.) Inputs to the THC in both modes result in movement within the Orbiter Body Axis Coordinate System. Because almost all tasks begin by moving the unloaded arm, the ORB UNL mode is the first mode learned by the student.



Advance Organizer

ORBITER UNLOADED MODE

"Orbiter Unloaded mode (ORB UNL) is used primarily to move the arm when no payload is attached, while the Orbiter Loaded mode is used primarily for berthing and unberthing operations. The major difference between these two modes is the payload ID. PL ID 0 (zero) is used for ORB UNL mode, which places the POR in the tip of the EE, and PL ID 1 is used for ORB LD mode, which usually defines the POR as a point within the payload. Inputs to the THC in both modes result in movement within the Orbiter body axis coordinate system. Because almost all tasks begin by moving the unloaded arm, the ORB UNL mode is the first mode you will learn."

1.3.1.1 THC INPUT/RESPONSE

Task 1.3.1.1a: Exercise: Translational Movement in ORB UNL Mode using BACS

The arm is in ORB UNL mode. The POR is highlighted; this visual aid is not removed during this exercise. Both the mode and the coordinate system are clearly labeled on the screen as before. The student sees a graphic depiction of the full arm with end effector pointed into the payload bay. One window contains a view from camera C, one window has a gods-eye view, one window displays panel A8, and one window uses the view from the shuttle aft window. The student uses the THC to move the arm as long as desired. If a reach limit or software stop is encountered, the student is so informed, and the simulation is re-started. The purpose is to learn how the arm moves in this mode and coordinate system. If the student hits a singularity or reach limit, the task resets itself and the student can start over.

Visual cues: The POR is highlighted. Also, a vector graphic depicting the XYZ axes in the Orbiter Body Axis Coordinate system is displayed. These cues are not faded in this exercise.

Advance Organizer:

"Understanding the movements that result from the various hand-controller inputs in each mode is one of the key skills for controlling the arm. The purpose of this exercise is for you to familiarize yourself with the translational movement of the arm in Orbiter Unloaded (ORB UNL) mode. In ORB UNL mode, PL ID 0 is used. You may recall that the POR in this mode is the tip of the end effector. Rate commands through the THC will result in motions of the POR that are parallel to the Orbiter body axis coordinate system.

"In this task, simply move the arm around using only the THC. Watch how the arm responds. Note how the coordinates change. When you feel comfortable with your understanding of translational movement in ORB UNL mode, press the NEXT TASK button for an exercise.

"This exercise is not evaluated."

Levels of Complexity: NA

Evaluation: NA

Advancement criteria: Student-controlled. A NEXT TASK button is available that moves the student to the next part task.

#### **Task 1.3.1.1b: Translational Movement In ORB UNL Mode using BACS**

The student sees the same views and windows described in Task 1.3.1.1a except that the aft window view is replaced by the performance indicator. In this task, the student must perform short fly-to's that only require use of the THC. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

Advance Organizer:

"Orbiter Unloaded mode (ORB UNL) is used primarily to move the arm when no payload is attached. Understanding the movements that result from the various hand-controller inputs in each mode is one of the key skills for controlling the arm. The purpose of this task is to demonstrate an understanding of translational arm movement in ORB UNL mode using only the THC.

"The POR in this mode is located on the tip of the end effector. Rate commands through the THC will result in motions of the POR that are parallel to the Orbiter body axis coordinate system.

"This task requires that you only use the THC to move the arm to the goal position. In the first two levels of complexity, a wire-frame "ghost arm" is used as the target, and will show the location and orientation of the goal position so that you can compare the visual position with the coordinate data.

"The task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three THC inputs simultaneously. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position, and of time. Accuracy is weighted higher than efficiency; Time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 15 seconds.

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints for each LOC.

### 1.3.1.2 RHC INPUT/RESPONSE

Arm operators use the Orbiter Rotation Axis Coordinate System (RACS) primarily as an aid to understand better the arm reconfiguration that results from commands input into the RHC and as an aid in understanding the attitude of the EE. The RACS is used in conjunction with the view from all cameras and windows.

**Task 1.3.1.2a: Exercise: Rotational Movement in ORB UNL Mode using RACS**

The full arm is shown with the End Effector pointing towards the ohms pods. One window contains a view from Camera C, one shows a gods-eye view, one contains panel A8, and one shows the view from the shuttle aft window. The student uses the RHC to move the arm in ORB UNL mode and observes the effects.

The student is reminded that inputs to the RHC cause the EE to rotate *around* the POR (i.e., there is no movement of the POR). If a reach limit or software stop is encountered, the student is so informed, and the simulation is re-started.

**Advance Organizer**

The purpose of this exercise is for you to familiarize yourself with the rotational movement of the arm in ORB UNL mode. Rate commands input through the RHC cause the EE to rotate around the POR (i.e., there is no movement of the POR). Arm operators use the Orbiter rotation axis coordinate system (RACS) in order to understand the arm reconfiguration that results from RHC commands, and to understand the attitude of the EE.

"In this task, simply move the arm around using only the RHC. Watch how the arm responds. Note how the coordinates change. When you feel comfortable with your understanding of rotational movement in ORB UNL mode, press the NEXT TASK button for an exercise.

"This exercise is not evaluated."

**Performance indicator: NA**

**Levels of Complexity: NA**

**Evaluation: NA**

**Advancement criteria: Student-controlled. A "NEXT TASK" button is available that moves the student to the next part task.**

**Task 1.3.1.2b: Rotational Movement in ORB UNL Mode using RACS**

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The coordinate system and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that only require use of the RHC. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

Advance Organizer

"The purpose of this task is to demonstrate an understanding of rotational arm movements in ORB UNL mode using only the RHC.

"The POR in this mode is located on the tip of the end effector. Rate commands through the RHC will result in motions of the POR that are parallel to the Orbiter rotation axis coordinate system.

"This task requires that you only use the RHC to move the arm to the goal position. In the first two levels of complexity, a wire-frame "ghost arm" is used as the target, and will show the location and orientation of the goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

HELP section: As above

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) Vector visual aid is removed and the arm moves in a direction that requires two RHC inputs simultaneously
- (3) The arm moves in a direction that requires three RHC inputs simultaneously. Godseye view is removed and replaced by a button.

- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 15 seconds.

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

### 1.3.1.3 INTEGRATION OF THC AND RHC INPUT/RESPONSE, ORB UNL

#### Task 1.3.1.3a: Integrating THC and RHC Movement in ORB UNL Mode using BACS and RACS

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The coordinate systems (BACS and RACS) and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that require use of both hand controllers. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

#### Advance Organizer

"The purpose of this task is to demonstrate an understanding of both translational and rotational arm movements in ORB UNL mode. Using both hand controllers simultaneously is the desired method of moving the arm to goal positions and attitudes.

"This task requires that you use both hand controllers (total: six degrees of freedom) to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time

varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

**Levels of Complexity:**

- (1) The arm begins in different positions/attitudes
- (2) Vector visual aid is removed and the arm moves in a direction that requires two RHC or THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC or RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

**Evaluation:** Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. **Efficiency:** Student should move the arm within 115% of the shortest possible path; both hand controllers should be used simultaneously for at least 25% of the task. When time is used, student should complete the task within 15 seconds.

**Advancement criteria:** Student must get five consecutive correct trials without errors and within time constraints per LOC.

### **1.3.1 ORBITER LOADED MODE**

#### **Advance Organizer**

#### **ORBITER LOADED MODE**

"Orbiter Loaded mode (ORB LD) is used primarily for berthing and unberthing operations. PLID 1 is used for ORB LD mode, which usually defines the POR as a point within the payload. Like ORB UNL, inputs to the THC result in movement within the Orbiter body axis coordinate system.

#### 1.3.1.4 THC INPUT/RESPONSE

(Note: 1.3.1.4a, an unevaluated free-play exercise for using the THC in ORB LD mode, is not developed at this time.)

##### Task 1.3.1.4b: Translational Movement in ORB LD Mode using BACS

The student sees the same views and windows described in Task 1.3.1.1a except that the aft window view is replaced by the performance indicator. In this task, the student must perform short fly-to's that only require use of the THC. The fly-to target position is indicated by a ghost arm image. Specific payloads are picked randomly. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

##### Advance Organizer:

\*Orbiter Loaded mode (ORB LD) is used primarily for berthing and unberthing operations. Understanding the movements that result from the various hand-controller inputs in each mode is one of the key skills for controlling the arm. The purpose of this task is to demonstrate an understanding of translational arm movements in ORB LD mode using only the THC.

\*The POR in this mode is located inside the payload. Rate commands through the THC will result in motions of the POR that are parallel to the Orbiter body axis coordinate system regardless of the orientation of the end effector.

\*This task requires that you use only the THC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

\*This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

\*This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm into the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.



"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payload is used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 120-240 seconds (a global editable parameter).

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints for each LOC.

### 1.3.1.5 RHC INPUT/RESPONSE

Arm operators use the Orbiter Rotation Axis Coordinate System (RACS) primarily as an aid to understand better the arm reconfiguration that results from commands input into the RHC and as an aid in understanding the attitude of the EE. The RACS is used in conjunction with the view from all cameras and windows.

(Note: 1.3.1.5a, an unevaluated free-play exercise for using the RHC in ORB LD mode, is not developed at this time.)

#### Task 1.3.1.5b: Rotational Movement in ORB LD Mode using RACS

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The coordinate system and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that only require use of the RHC. The fly-to target position is indicated by a ghost arm image. Specific payloads are picked randomly. If the student hits a

singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

**Advance Organizer**

"The purpose of this task is to demonstrate an understanding of rotational arm movements in ORB LD mode using only the RHC.

"The POR in this mode is located inside the payload. Rate commands through the RHC will result in motions of the POR that are parallel to the Orbiter rotation axis coordinate system regardless of the orientation of the end effector.

"This task requires that you use only the RHC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm into the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

**HELP section: As above**

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) Vector visual aid is removed and the arm moves in a direction that requires two RHC inputs simultaneously
- (3) The arm moves in a direction that requires three RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payloads are used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 120-240 seconds (a global editable parameter).

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

### 1.3.1.6 INTEGRATION OF THC AND RHC INPUT/RESPONSE, ORB LD MODE

#### Task 1.3.1.6a: Integrating THC and RHC Movement in ORB LD Mode using BACS and RACS

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The coordinate systems (BACS and RACS) and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that require use of both hand controllers. The fly-to target position is indicated by a ghost arm image. Specific payloads are picked randomly. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the DONE button.

#### Advance Organizer

"The purpose of this task is to demonstrate an understanding of both translational and rotational arm movements in ORB LD mode. Using both hand controllers simultaneously is the desired method of moving the arm to goal positions and attitudes.

"This task requires that you use both hand controllers (total: six degrees of freedom) to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: Imagine you are grasping the POR (as seen in the aft window view) and not the hand controllers. Move your hands as you would the arm in order to achieve the goal position. Also remember that the RMS has proportional hand controllers. This means that small deflections produce slower movement, larger deflections produce faster movement. Multiple axis inputs are allowed and desirable where applicable."

HELP section: As above

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) Vector visual aid is removed and the arm moves in a direction that requires two RHC or THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC or RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payloads are used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path; both hand controllers should be used simultaneously for at least 25% of the task. When time is used, student should complete the task within 120-240 seconds (a global editable parameter).

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

### 1.3.2 END EFFECTOR MODE

End Effector mode (END EFF) is primarily used for grappling and ungrappling procedures. The POR is located in the tip of the EE and the EE coordinate system is used. This coordinate system should be taught while teaching arm operation using END EFF.

The EE coordinate system is primarily used as an aid to help arm operators understand the arm (EE) movement that results from inputs to the hand-controllers. By providing a more intuitive perspective, it allows operators to visualize arm movements (directional and arm reconfiguration) prior to inputting commands to both the THC and RHC. This coordinate system is always used in conjunction with the view from the wrist camera. However, it is necessary to keep in mind that the wrist camera may be used for visual information in other coordinate systems.

#### 1.3.2.1 THC INPUT/ARM RESPONSE

##### Task 1.3.2.1a: Exercise: Translational Movement in END EFF Mode using ECS

The arm is placed in the same configuration as Task 1.3.1.2a; the windows are similar. The point of resolution (POR) is highlighted; this cue is not faded during this exercise. The coordinate system and mode are prominently labeled on the screen. The student uses the THC to move the arm and observes the effects. If a reach limit or software stop is encountered, the student is so informed, and the simulation is re-started.

##### Advance Organizer

"The purpose of this exercise is for you to familiarize yourself with the translational movement of the arm in end effector mode (END EFF). END EFF is primarily used for grappling and ungrappling procedures. The POR is located in the tip of the EE and the EE coordinate system is used. In END EFF, the arm functions in an intuitive manner (i.e., like your own arm). For this reason, operators can visualize arm movements more easily prior to inputting commands to either or both hand controllers. The EE coordinate system is always used in conjunction with the view from the wrist camera.

"In this task, simply move the arm around using only the THC. Watch how the arm responds. Note how the coordinates change. When you feel comfortable with your understanding of translational movement in END EFF, press the NEXT TASK button for an exercise.

"This exercise is not evaluated."

- . Levels of Complexity: NA
- . Performance indicator: NA

Evaluation: NA

Advancement criteria: Student-controlled. A "NEXT TASK" button is available that moves the student to the next part task.

#### **Task 1.3.2.1b: Translational Movement in END EFF Mode using ECS**

The student is given the same views and windows and visual aid as described in Task 1.3.2.1a except that the performance indicator replaces the aft shuttle window view. The coordinate system and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that only require use of the THC. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

Advance Organizer

"The purpose of this task is to demonstrate an understanding of translational arm movements in END EFF mode using only the THC.

"The POR in this mode is located on the tip of the end effector and the end effector coordinate system is used. In END EFF, the arm functions in an intuitive manner (i.e., like your own arm). For this reason, operators can visualize arm movements more easily prior to inputting commands to either or both hand controllers. The end effector coordinate system is always used in conjunction with the view from the wrist camera.

"This task requires that you only use the THC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm into the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The END EFF coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the END EFF coordinate system. Positive X is always in the direction of the END EFF camera; positive Z is down in the END EFF camera view and Y completes the right-hand rule."

**Levels of Complexity:**

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

**Evaluation:** Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. **Efficiency:** Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 15 seconds.

**Advancement criteria:** Student must get five consecutive correct trials without errors and within time constraints per LOC.

### **1.3.2.2RHC INPUT/RESPONSE**

#### **Task 1.3.2.2a: Exercise: Rotational Movement in END EFF Mode using RACS**

The student is given a graphic depiction of the full arm with the End Effector pointing into the payload bay. The POR is highlighted and vectors indicating the three axes of the EE coordinate system emanate from the POR; also, a graphic showing the three axes of the RACS is shown. These cues are not faded during this exercise. The coordinate system and mode are prominently displayed.

In addition to the normal P2T2 aft window, a window showing the view from the wrist camera and a window using the gods-eye view are present. The fourth P2T2 window is used for panel A8. The student uses the RHC to move the arm and observes the effects. If a reach limit or software stop is encountered, the student is so informed, and the simulation is re-started.

Advance Organizer

"In this exercise, you can familiarize yourself with the rotational movement of the arm in END EFF.

"Simply move the arm around using only the RHC. Watch how the arm responds. Note how the coordinates change.

When you feel comfortable with your understanding of rotational movement in END EFF, press the NEXT TASK button for a related task.

"This exercise is not evaluated."

Levels of Complexity: NA

Evaluation: NA

Advancement criteria: Student-controlled. A "NEXT TASK" button is available that moves the student to the next part task.

**Task 1.3.2.2b: Rotational Movement in END EFF Mode using RACS**

The student is given the same views and windows as described in Task 1.3.2.2a, except that the performance indicator/feedback is shown instead of the aft window view. The coordinate system and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that only require use of the RHC. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

Advance Organizer

"The purpose of this task is to demonstrate an understanding of rotational arm movements in END EFF mode using only the RHC.

"The POR in this mode is located on the tip of the end effector and the end effector coordinate system is used. In END EFF, the arm functions in an intuitive manner (i.e., like your own arm). For this reason, operators can visualize arm movements more easily prior to inputting commands to either or both hand controllers. The end effector coordinate system is always used in conjunction with the view from the wrist camera.

"This task requires that you only use the RHC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.



"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The END EFF coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the END EFF coordinate system. Positive X is always in the direction of the END EFF camera; positive Z is down in the END EFF camera view and Y completes the right-hand rule."

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two RHC inputs simultaneously. (Unless LOC 2 is implemented, remove visual aids except gods-eye at this LOC.)
- (3) The arm moves in a direction that requires three RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 15 seconds.

Advancement criteria: Student must get five consecutive correct trials without errors per LOC.

### 1.3.2.3 INTEGRATION OF THC AND RHC INPUT/RESPONSE, END EFF MODE

#### Task 1.3.2.3a: Integrating THC and RHC Movement in END EFF Mode

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The coordinate systems (BACS and RACS) and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that require use of both hand controllers. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

##### Advance Organizer

"The purpose of this task is to demonstrate an understanding of both translational and rotational arm movements in END EFF mode. Using both hand controllers simultaneously is the desired method of moving the arm to goal positions and attitudes.

"This task requires that you use both hand controllers (total: six degrees of freedom) to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The END EFF coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the END EFF coordinate system. Positive X is always in the direction of the END EFF camera; positive Z is down in the END EFF camera view and Y completes the right-hand rule."

HELP section: As above

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two RHC or THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC or RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. Hand controllers should be used simultaneously at least 25% of the time. When time is used, student should complete the task within 15 seconds.

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

### **1.3.3 PAYLOAD MODE AND PAYLOAD COORDINATE SYSTEM**

The Payload Mode of operation is used by the operator when it is desirable to move the payload in directions that correspond to the orientation of the payload. The POR is defined to be a point in space, but is usually located within the payload.

The Payload Coordinate System is primarily used as an aid to help arm operators understand the arm (payload) movement that results from inputs to the hand controllers. By providing a more intuitive perspective, it allows operators to visualize arm movements (directional and arm reconfiguration) prior to inputting commands.

#### **1.3.3.1 THC INPUT/RESPONSE**

##### **Task 1.3.3.1a: Exercise: Translational Movement in Payload Mode using PCS**

The student sees a graphic depiction of the full arm with the midline of the End Effector parallel to the Z axis in BACS. A payload is attached to the EE. One window shows a view from the wrist camera and one shows a gods-eye view. In addition, the normal P2T2 aft window view and panel A8 are displayed. The POR is highlighted (located in the payload) and axes in the Payload Coordinate System (PCS) are displayed (emanating from the payload POR); these aids are not faded during this exercise. The student uses the THC to move the arm and observes the effects. If a reach limit or software stop is encountered, the student is so

informed, and the simulation is re-started. The coordinate system and mode are prominently displayed on the screen. Specific payloads are picked randomly.

. Advance Organizer

"The purpose of this exercise is for you to familiarize yourself with translational movement of the arm in Payload mode. Payload mode is used by the operator when it is desirable to move the payload in directions that correspond to the orientation of the payload. The POR is defined to be a point in space, but is usually located within the payload.

"In this exercise, the POR is highlighted (located in the payload) and axes in the Payload Coordinate System (PCS) are displayed (emanating from the payload POR).

"Simply move the arm around using only the THC. Watch how the arm responds. Note how the coordinates change. When you feel comfortable with your understanding of translational movement in Payload mode, press the NEXT TASK button for an exercise.

"This exercise is not evaluated."

. Levels of Complexity: NA

. Evaluation: NA

. Advancement criteria: Student-controlled. A "NEXT TASK" button is available that moves the student to the next part task.

**Task 1.3.3.1b: Translational Movement in Payload Mode using PCS**

The student is given the same views and windows as described in Task 1.3.3.1a, except that the aft window view is replaced by the performance indicator and feedback. The coordinate system and mode are prominently displayed on the screen. In this task, the student must perform short fly-to's that only require use of the THC. The fly-to target position is indicated by a ghost arm image. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button. Specific payloads are picked randomly.

. Advance Organizer

"The PAYLOAD Mode of operation is used by the operator when it is desirable to move the payload in directions that correspond to the orientation of the payload. The purpose of this task is to demonstrate an understanding of translational arm movements in PAYLOAD mode using only the THC in the Payload coordinate system.

"The POR is defined to be a point in space, but is usually located within the payload. The Payload Coordinate system is primarily used as an aid to help arm operators understand the arm (payload) movement that results from inputs to the hand controllers. By providing a more intuitive perspective, it allows operators to visualize arm movements (directional and arm reconfiguration) prior to inputting commands.

"This task requires that you use the THC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data. The axes in the Payload Coordinate System are displayed and emanate from the payload POR. The X axis is represented in blue, the Y axis in yellow, and the Z axis in red.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm into the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The Payload coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the Payload Coordinate System. Positive X, Y, Z are in the direction defined by the payload ID."

HELP section: As above

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two THC inputs simultaneously.
- (3) The arm moves in a direction that requires three THC inputs simultaneously. Godseye view is removed and replaced by a button.

- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payloads are used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 120-240 seconds (a global editable parameter).

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

### 1.3.3.2RHC INPUT/RESPONSE

#### Task 1.3.3.2a: Exercise: Rotational Movement in Payload Mode using PCS

The student is given a graphic depiction of the full arm with the midline of the End Effector parallel to the Z axis in the Orbiter Body Axis Coordinate system. A payload is attached to the EE. The POR (located within the payload) is highlighted with three vectors representing the PCS axes emanating from the POR; this visual cue is not faded during the exercise. The student sees a view from the wrist camera, a gods-eye view of the Orbiter, and the normal P2T2 aft window and panel A8. The coordinate system and mode are prominently displayed. The student uses the RHC to move the arm and observes the effects. If a reach limit or software stop is encountered, the student is so informed, and the simulation is re-started. Specific payloads are picked randomly.

#### Advance Organizer

"The purpose of this exercise is for you to familiarize yourself with the rotational movement of the arm in Payload mode.

"In this exercise, the POR (located within the payload) is highlighted with three vectors representing the PCS axes emanating from the POR.

"Simply move the arm around using only the RHC. Watch how the arm responds. Note how the coordinates change. When you feel comfortable with your understanding of rotational movement in Payload mode, press the NEXT TASK button for an exercise.

"This exercise is not evaluated."

Levels of Complexity: NA

Evaluation: NA

Advancement criteria: Student-controlled. A "NEXT TASK" button is available that moves the student to the next part task.

### **Task 1.3.3.2b: Rotational Movement In Payload Mode using PCS**

The student is given the same views and windows as described in Task 1.3.3.2a. In this task, the student must perform short fly-to's that only require use of the RHC. The fly-to target position is indicated by a ghost arm image. When done, the student presses the NEXT TASK button. The coordinate system and mode are prominently displayed on the screen. If the student hits a singularity or reach limit, the trial will reset and the student can start over. Specific payloads are picked randomly.

#### **Advance Organizer**

"The purpose of this task is to demonstrate an understanding of rotational arm movements in PAYLOAD mode using only the RHC in the Payload coordinate system.

"The POR is defined to be a point in space, but is usually located within the payload. The Payload Coordinate system is primarily used as an aid to help arm operators understand the arm (payload) movement that results from inputs to the hand controllers. By providing a more intuitive perspective, it allows operators to visualize arm movements (directional and arm reconfiguration) prior to inputting commands.

"This task requires that you use the RHC to move the real arm to the goal position. In the first two levels of complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data. The axes in the Payload Coordinate System are displayed and emanate from the payload POR. The X axis is represented in blue, the Y axis in yellow, and the Z axis in red.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The Payload coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the Payload Coordinate System. Positive X, Y, Z are in the direction defined by the payload ID."

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two RHC inputs simultaneously.
- (3) The arm moves in a direction that requires three RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payloads are used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, student should complete the task within 120-240 seconds.

Advancement criteria: Student must get five consecutive correct trials without errors within time constraints per LOC.

### 1.3.3.3 INTEGRATION OF THC AND RHC INPUT/RESPONSE, PAYLOAD MODE

#### Task 1.3.3.3a: Integrating THC and RHC Movement in Payload Mode

The student is given the same views and windows as described in Task 1.3.1.2a except that the aft window view is replaced by the performance indicator. The same visual aids are also used. The payload coordinate system and payload mode are prominently labeled on the screen. In this task, the student must perform short fly-to's that require use of both hand controllers. The fly-to target position is indicated by a ghost arm image. Specific payloads are picked randomly. If the student hits a singularity or reach limit, the trial will reset and the student can start over. When done, the student presses the NEXT TASK button.

#### Advance Organizer

"The purpose of this task is to demonstrate an understanding of both translational and rotational arm movements in PAYLOAD mode. Using both hand controllers simultaneously is the desired method of moving the arm to goal positions and attitudes.

"This task requires that you use both hand controllers (total: six degrees of freedom) to move the real arm to the goal position. In the first two levels of



complexity, a wire frame "ghost arm" is used as the target, and will show the location and orientation of that goal position so that you can compare the visual position with the coordinate data. The axes in the Payload Coordinate System are displayed and emanate from the payload POR. The X axis is represented in blue, the Y axis in yellow, and the Z axis in red.

"This task is divided into four levels of complexity. The first three ask that you move the arm in a direction that requires first one, then two, and three inputs simultaneously for each hand controller. The fourth level of complexity takes away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy and efficiency. Accuracy is measured by the final proximity of the real arm to the goal position; Efficiency is measured by your ability to move the arm in a straight line to the goal position (single-axis inputs and flying boxes will rate low on the efficiency rating), and of time. Accuracy is weighted higher than efficiency; time varies per level of complexity and begins when you move the hand controllers.

"When you have finished the task, use the mouse cursor to press the "Done" button.

"Hint: The Payload coordinate system is a rotating coordinate system. Hand-controller inputs to move in a particular direction with respect to the orbiter are dependent on the current orientation of the Payload Coordinate System. Positive X, Y, Z are in the direction defined by the payload ID."

HELP section: As above

Levels of Complexity:

- (1) The arm begins in different positions/attitudes
- (2) The arm moves in a direction that requires two RHC or THC inputs simultaneously
- (3) The arm moves in a direction that requires three THC or RHC inputs simultaneously. Godseye view is removed and replaced by a button.
- (4) Slightly longer fly-to's are used; ghost arm is removed and replaced by a button; larger payloads are used.

Evaluation: Accuracy is judged by how well the arm is lined up with the ghost arm. Student is allowed 10% discrepancy. Efficiency: Student should move the arm within 115% of the shortest possible path. Student should use handcontrollers simultaneously at least 25% of the time. When time is used, student should complete the task within 120-240 seconds (global editable parameter).

Advancement criteria: Student must get five consecutive correct trials without errors and within time constraints per LOC.

#### 1.4. VISUALIZATION SKILLS (ARM CONFIGURATION/EE POSITION-AND-ATTITUDE/PAYLOAD POSITION-AND-ATTITUDE)

Good visualization skills involve more than the ability to identify the location of points in space. They also include the ability to create a mental image of (1) both the movement of the POR (translational) and of the EE around the POR (rotational) as viewed from the wrist camera; and (2) the ability to image the arm movement (reconfiguration) that results from both THC and RHC inputs. The first visualizations relate to the coordinate system involved, and the second corresponds to the EE attitude in relationship to the rest of the arm.

The "gods-eye" view of the Orbiter is a useful perspective during arm manipulation, but this view is not "realistic." NASA instructors believe this view is useful during arm manipulation nevertheless, because it provides a very good perspective of arm configuration and movement; but they are reluctant to use it in training because it is not available in orbit.

However, experienced arm operators employ mental images during arm operation in addition to the actual views available from Orbiter cameras and windows. Astronauts can reproduce these mental images because they have seen the various arm configurations from different perspectives over and over again throughout their training. The ability to reproduce the gods-eye view mentally is a useful skill. If the student has experienced that view (through practice at the P2T2) in a multitude of situations, he/she could imagine it more easily. This experience would provide the arm operator with a tool he/she does not currently have.

Some visualization skills can be taught on the P2T2 simulator, while others cannot. One shortcoming is that some positions and attitudes of the EE are difficult to ascertain visually, since the screen is actually 2-dimensional. During later part tasks, students are taught to use the cameras effectively to help overcome this problem, which is also a problem in the real world of RMS operations. However, in this section, care is taken to select tasks that provide the arm operator with a sufficiently clear view of the EE position and attitude.

### Task 1.4a: Visual Recognition of Arm Positions/Attitudes

Because P2T2 cannot display arms in different positions on the same screen, three different screens are used for this task, each depicting a different arm configuration. Three windows on each screen show different views of the arm, and the fourth window is used for the performance indicator and feedback. Also in the fourth window, the student is given a target EE position and attitude (i.e., a set of XYZ coordinates in BACS, as well as pitch, yaw, and roll coordinates in RACS. Panel A8, or at least the readouts on Panel A8, must be covered.

The student presses a "See another choice" button to view (cycle through) the various possibilities. The student must then select the arm that meets the target coordinates by pressing a button titled "Select third arm," "Select first arm," "Select second arm." (Each screen should also be labeled accordingly so that it is always clear which is the "first" arm, or whatever.) No review is allowed.

When the timer is used: The student can view each screen for 30 [60] seconds; then the screen changes. Timer begins after the screen plots. When the third screen plots, student has 30 [60] seconds to view the arm and make a selection. (When the timer is not used, no timer limits are imposed and a button should be available to change views.)

#### Advance Organizer:

"Purpose: Develop visualization skills related to judging the relative position and attitude of the arm. Visualization skill is an important part of flying the RMS. You must know where you are as well as where you are going.

Task: You are given a set of position-and-attitude coordinates. Three different arm configurations will be displayed on three different screens. You must choose the arm that represents the target coordinates. Remember what you see on each screen, because you cannot review previous screens.

Evaluation criteria: Correct/incorrect is the only evaluation.

Hint: Remember what you have learned when performing previous tasks."

#### Levels of Complexity:

- (1) Use coordinates that are both obviously correct for the target arm, and obviously NOT correct for the distractors. Start with arm position/attitudes that are simpler to assess.
- (2) Use coordinates that are more possible for the distractors, and also arm positions that are more difficult to assess. (Keep the attitude the same for each arm.)

- (3) Use coordinates that are more possible for the distractors, and arm attitudes that are more difficult to assess. (Pitch around Y, yaw around the new Z, roll around X.)

. Evaluation: This task is an evaluation of accuracy, based on number of correct vs. incorrect responses. When the timer is used, an efficiency measurement is based on the student responding within 30 seconds for LOC 1, and 60 seconds for LOC 2.

. Advancement criteria: Student is correct on 5 consecutive trials per LOC without error.

## Part Tasks for Manipulation Skills

### 2. THC AND RHC MOTOR SKILLS

The THC and RHC motor skills necessary for effective arm manipulation involve the ability to translate a desired arm movement into a THC or RHC input. These skills should be automated.

Arm response to THC and RHC inputs varies depending on the mode of operation. Thus, it is not sufficient to have only motor skills related to the hand controllers. The arm operator must also anticipate the result of inputs in each of the various modes. Note that THC and RHC inputs to the P2T2 bring about the same arm response as an input from the hand controllers at the other simulators (e.g., SES).

LOCs: Except in a few places, a minimum of three levels of complexity (LOCs) will be implemented per task.

#### 2.1 THC AND RHC (PSYCHOMOTOR)

The first step in training manipulation skills is to help the student develop hand-eye coordination as well as experience with different modes. This is done first with one hand controller in two modes, then with the other, then both hand controllers simultaneously. The task begins with a relatively easy tracing job, then uses more complex objects to trace.

##### Advance Organizer

##### MANIPULATION SKILLS

"As you are now aware, arm response to THC and RHC inputs varies depending on the mode of operation. Thus, it is not sufficient to have only motor skills related to the hand controllers. You must also anticipate the result of inputs in each of the various modes.

"The next group of tasks are tracing tasks. You will manipulate the arm to trace certain objects such as the ohms pods. These tasks will help you develop hand-eye coordination as well as more experience with arm movement in different modes.

"In these tasks, you will work first with one hand controller in two modes (END EFF and ORB UNL), then with the other. Finally, you will use both hand controllers simultaneously."

## 2.2 THC CONTROL

### Task 2.2: Developing More Precise Arm Control using the THC

The EE is pointed at the bulkhead with the wrist camera as the primary source of visual information for both modes. The task uses END EFF mode, which fact is prominently displayed on the screen. The outline of the bulkhead is highlighted and cross hairs are depicted on the screen with the intersection of the cross hairs positioned on the outline of the bulkhead. (Care must be taken so that the bulkhead line does not require the arm to travel too close to structure.) The student traces the outline of the bulkhead with the cross hairs, traveling from left to right and then back using only the THC.

#### Advance Organizer:

"One purpose of this task is to begin to develop better hand-eye coordination when moving the arm. Another purpose is to give you more experience with various modes of operation. The task begins in END EFF mode, and later changes to ORB UNL.

"In this task, the wrist camera is used, which has cross hairs superimposed over the "lens." Your job is to use only the THC to move the arm and trace a line with the cross hairs.

"This task is evaluated for accuracy and efficiency. The acceptable tolerance for accuracy gets narrower as the task proceeds from one exercise to the next, and is a function of distance of the cross hairs from the trace line. Efficiency is a function of time, and is measured only if the timer is used.

"Arrows show the direction in which to move the arm; a bar at the end of the trajectory indicates the "goal point."

#### Levels of Complexity:

- (1) Envelope of acceptable performance = 2 feet; see below under "Evaluation"
- (2) Envelope of acceptable performance = 1 foot
- (3) Envelope of acceptable performance = 6 inches; add time requirement (for remaining LOCs as well)
- (4) The student traces the ohms pods/tail with 6-inch envelope
- (5) The above tasks (tracing the bulkhead, tracing the ohms pods) are repeated using ORB UNL mode and 1 foot envelope.

- (6) The above tasks (tracing the bulkhead, tracing the ohms pods) are repeated using ORB UNL mode and 6-inch envelope.

Evaluation: An "envelope" of acceptable performance is defined on each side of the bulkhead trace line. This corresponds to, first, two feet of clearance on each side, then one foot, and finally six inches. (These figures are relative to the actual Orbiter.) Also, time in minutes is kept as an order of complexity.

When the timer is used and time is a factor: for tracing the bulkhead, 4 minutes to go and return; for tracing the ohms pods including the tail: 8 minutes to go and return.

Advancement Criteria: Student performs three acceptable trials in each complexity level and for each mode without error within each set (e.g., the student performs three error-free trials in END EFF mode with the 2-foot envelope; then the student uses the one-foot envelope and makes some errors, but finally performs three trials without error and progresses to the 6-inch envelope, etc.)

## 2.3 RHC CONTROL

### Task 2.3: Developing More Precise Arm Control using the RHC

The EE is pointed at the bulkhead with the EE camera as the primary source of visual information. END EFF mode is used for this task, which fact is prominently displayed on the screen. The outline of the bulkhead is highlighted and cross hairs associated with the wrist camera are depicted on the screen with the intersection of the cross hairs positioned on the outline of the bulkhead. The student traces the outline of the bulkhead with the cross hairs, traveling from left to right and then back using only the RHC.

#### Advance Organizer:

"In this task, you will develop better hand-eye coordination when moving the arm, and gain more experience with various modes of operation. The task begins in END EFF mode, and later changes to ORB UNL.

"In this task, the wrist camera is used, which has cross hairs superimposed over the "lens." Your job is to use only the RHC to move the arm and trace a line with the cross hairs. The figure below shows the optimal relationship (most efficient trajectory) between the cross hairs and the trace line:

<figure of the optimal juxtaposition between cross hairs and trace line>

"This task is evaluated for accuracy and efficiency. The acceptable tolerance for accuracy gets narrower as the task proceeds from one exercise to the next, and is a function of distance of the cross hairs from

the trace line. Efficiency is a function of how smoothly you move the arm, and, when the timer is used, of time.

"Arrows show the direction in which to move the arm; a bar at the end of the trajectory indicates the "goal point." "

HINT buttons: If the student is not inputting 2-3 directions simultaneously, give the following hint:

"In order to perform this task in the most efficient manner, try to move the arm in 2-3 directions simultaneously at any given time.

"You might also imagine a light beam going from the EE to the bulkhead or ohms pod, with your hand as the EE. Try to move the light beam along the path you want to trace in a smooth fashion."

Levels of Complexity:

- (1) Envelope of acceptable performance is 2 feet
- (2) Envelope of acceptable performance is 1 foot
- (3) Envelope of acceptable performance is 6 inches; also, a time limit is used (retain through remaining tasks)
- (4) The student traces the ohms pods/tail
- (5) The above tasks (tracing the bulkhead, tracing the ohms pods) are repeated using ORB UNL mode.

Evaluation: An "envelope" of acceptable performance is defined on each side of the bulkhead trace as in task 2.2. Also, evaluate efficiency by how smooth the student's trajectory is: make sure the student is always inputting 2-3 directions simultaneously. When the timer is used, time counts as 1/3 of the efficiency measure.

Advancement Criteria: As for task 2.2.

## **2.4 INTEGRATED USE OF THE THC AND RHC (PSYCHOMOTOR)**

### **Task 2.4: Developing More Precise Arm Control using Both Hand Controllers Simultaneously**

In this task, the student integrates the use of the hand controllers by inputting commands into the THC and RHC simultaneously. END EFF mode is used for this task with the wrist camera as the primary source of visual information (top left window: view from the aft shuttle window; top right: wrist camera; bottom right: camera B; bottom left: panel A8). Cross hairs are placed in the center of the wrist camera view. The outline of the bulkhead is highlighted by a line. The student



must trace the outline of the bulkhead using both hand controllers while keeping the intersection of the cross hairs on the bulkhead line and the horizontal part of the cross hairs tangent to the line.

Advance Organizer:

"The purpose of this task is to help you develop better hand-eye coordination when moving the arm, and gain more experience with various modes of operation. The task begins in END EFF mode, and later changes to ORB UNL.

"In this task, the wrist camera is used, which has cross hairs superimposed over the "lens." Your job is to use both hand controllers simultaneously to move the arm and trace a line with the cross hairs. The figure below shows the optimal relationship (most efficient trajectory) between the cross hairs and the trace line:

<figure of the optimal juxtaposition between cross hairs and trace line>

"This task is evaluated for accuracy and efficiency. The acceptable tolerance for accuracy gets narrower as the task proceeds from one exercise to the next, and is a function of distance of the cross hairs from the trace line. Efficiency is a function of using both hand controllers simultaneously, and, when the timer is running, of time.

"Arrows show the direction in which to move the arm; a bar at the end of the trajectory indicates the "goal point."

Levels of Complexity:

- 1) Envelope of acceptable performance narrows
- (2) Envelope of acceptable performance narrows again
- (3) Time becomes a factor (i.e., when using 6-inch envelope, then retain through remaining tasks)
- (4) The student traces the ohms pods/tail
- (5) The above tasks (tracing the bulkhead, tracing the ohms pods) are repeated using ORB UNL mode.

Evaluation:

Accuracy:

- (1) THC use: An "envelope" of acceptable performance is defined on each side of the bulkhead trace as before (task 2.2).
- (2) RHC use: Sum of the absolute value of the angle changes between the horizontal cross hair and the object outline. Value of acceptable performance will have to be developed by using the task and generating initial criteria.
- (3) Some algorithm for integrating these evaluative measures must be developed, since they cannot simply be added (distance + angles)

Efficiency: Student should use the hand controllers simultaneously at least 50% of the time. When time is used, it counts as 1/3 the efficiency measure.

Advancement Criteria: Student performs three consecutive acceptable trials in each complexity level and for each mode without error for each LOC.

## Part Tasks for Monitoring Skills

### 3. MONITORING SKILL

The goal of monitoring is to ensure safe, accurate, and efficient performance of the RMS task. Monitoring requires the ability to obtain and use the most appropriate information during task performance: numerical data from panel A8 and visual information from the various cameras and windows. Monitoring also requires an awareness of what to look for at any given point in the task.

In order to heighten student awareness of the possibilities for and importance of cameras and camera angles, the camera used and how the particular view was derived (e.g., "Camera D, zoomed, panned left"; these descriptions should be more specific if such data is available and meaningful) are prominently labeled where pertinent, regardless of the particular task at hand. (This is in addition to the fact that windows are automatically labeled by the P2T2 according to camera used; however, these labels are small and likely to be ignored.)

LOCs: Except in a few places, a minimum of three levels of complexity (LOCs) will be implemented per task.

#### Advance Organizer

#### MONITORING ARM MOVEMENT

"The goal of monitoring is to ensure safe, accurate, and efficient performance of the RMS task. Monitoring requires the ability to obtain and use the most appropriate information during task performance: numerical data from panel A8 and visual information from the various cameras and windows. Monitoring also requires an awareness of what to look for at any given point in the task.

"In this section on monitoring skills, you will learn to recognize and deal with singularities, reach limits, and software stops. You will also begin to be aware of the importance of using and adjusting the available cameras to get the optimal view for monitoring arm operation."

#### 3.1 SINGULARITY AVOIDANCE

When the arm is in a singularity, Orbiter software automatically reconfigures the arm to resolve the problem. This creates the possibility of collision with parts of the Orbiter or payload during reconfiguration; it also means time will be spent waiting for the reconfiguration to complete, thus possibly impacting mission schedules; and finally, unless the Orbiter is placed in free drift during resolution of the singularity, the arm itself can be damaged. For these reasons, it is important to avoid placing the arm in a singularity.

Students have learned what singularities are in core training (knowledge). To avoid singularities, the student must know what the various singularities look like in order to

recognize when the arm configuration is close to a singularity. Three basic singularities are possible.

**Advance Organizer**

**SINGULARITY RECOGNITION AND AVOIDANCE**

"A singularity occurs when the arm is driven into a configuration where one or more degrees of freedom are lost. There are three possible singularities: wrist gimbal lock, shoulder yaw, and planer pitch joint. (Refer to the Payload Deployment and Retrieval System Overview Workbook for details on singularities.)

"When the arm is in a singularity, Orbiter software automatically reconfigures the arm to resolve the problem. This creates the possibility of collision with parts of the Orbiter or payload during reconfiguration; it also means time will be spent waiting for the reconfiguration to complete, thus possibly impacting mission schedules; and finally, unless the Orbiter is placed in free drift during resolution of the singularity, the arm itself can be damaged. For these reasons, it is important to avoid placing the arm in a singularity."

**Task 3.1a: Recognizing Singularities**

Because the P2T2 cannot display arms in different positions on the same screen, and because panel A8 indicates whether the arm is in a singularity, three different screens are used for this task, each depicting a different arm configuration. On one of the screens, the arm demonstrates a shoulder yaw singularity; the other two do not have singularities of any kind. Three windows on each screen show the arm, and the fourth window is used for the performance indicator and feedback.

Three buttons are available for viewing the arm ("See Arm 1," "See Arm 2," "See Arm 3") and three buttons for selecting the arm that has a singularity ("Select Arm 1," "Select Arm 2," "Select Arm 3"). The student cannot use Spec 96 in this exercise except for wrist singularities.

**Advance Organizer:**

"A singularity occurs when the arm is driven into a configuration where one or more degrees of freedom are lost. There are three possible singularities: wrist gimbal lock, shoulder yaw and planer pitch joint. (Refer to the Payload Deployment and Retrieval System Overview Workbook for details on singularities.)

"When the arm is in a singularity, Orbiter software automatically reconfigures the arm to resolve the problem. This creates the possibility of collision with parts of the Orbiter or payload during reconfiguration; it also means time will be spent waiting for the reconfiguration to complete, thus possibly impacting mission schedules; and finally, unless the Orbiter is

placed in free drift during resolution of a singularity, the arm itself can be damaged. For these reasons, it is important to avoid placing the arm in a singularity.

"In this task you are to recognize arm configurations that result in a singularity. Identification of singularities will help you avoid such configurations while flying.

"You will see three views of the same arm configuration in each window of a screen; there are three screens with different arm configurations on them - one of which is a singularity. You must choose the arm that is in a singularity.

"This task is evaluated on whether your choice is correct. A timer is used to limit the amount of time you have to view the configurations and make a decision.

"Hint: There are only three possible singularities: Gimbal Lock is associated with the wrist yaw joint, and occurs when the wrist yaw joint is at plus or minus 90 degrees. A rotational degree of freedom (DOF) is lost. Shoulder Yaw occurs when the wrist yaw joint is directly over the shoulder yaw axis. The DOF lost is the ability to translate perpendicular to the plane containing the booms of the arm. Planer Pitch Joint occurs when the wrist pitch, elbow pitch, and shoulder pitch joints are collinear. The DOF lost is the ability to translate away from the shoulder joints."

HINT buttons: An editable number of buttons (currently three) are available that present a definition of singularities.

Hint text: "The shoulder yaw singularity occurs when the wrist yaw joint is within three feet of the shoulder yaw joint axis. The arm in this configuration experiences restricted motion that is normal to the pitch plane. Translation commands parallel to the Orbiter Y axis could be delayed as the shoulder yaw and wrist yaw joints are reconfigured.

The wrist yaw singularity, sometimes called gimbal lock, occurs when the wrist yaw joint is within 15 degrees of +90 or -90. This creates a configuration where wrist pitch and wrist yaw are parallel and requires arm reconfiguration when roll about the pitch plane is commanded.

The planer pitch joint singularity occurs when the three pitch joints are in a line, or more specifically, when the elbow pitch joint is at or above -7.6 degrees. Any degree of freedom that requires the arm to get longer is not available since the arm is essentially at its maximum length. This situation is seen anytime the arm is at or near its cradled position.

Levels of Complexity:

- (1) As above
- (2) Wrist yaw singularities are used
- (3) Planar pitch singularities are used
- (4) Singularities are all represented in random order; some non-singularities (distractors) are almost singularities

Evaluation: Accuracy is measured by number correct/incorrect. In addition, when the timer is used, the student has 30 seconds to make a selection.

Advancement Criteria: For each LOC, student must get three consecutive trials correct without error within time constraints.

**Task 3.1b: Visualization of and Driving the Arm into Singularities**

In this task, the student is given a series of arm configurations, one at a time. Panel A8 is used and one window is used for feedback. The other two windows show optimal views of a single arm configuration that is close to but not in a singularity. In each trial, the student must identify (1) the type of singularity (shoulder yaw, wrist yaw, or planar pitch) that is most likely to occur from the given configuration, and (2) what THC and/or RHC inputs would result in that singularity (e.g., positive roll). These activities are performed by using the mouse to move a cursor over selections in the feedback window and clicking the mouse button. A simple "Correct" or "Wrong" appears, and the student must keep trying (cannot proceed) until correct. Make sure student does NOT use Spec 96 during this exercise.

The student must then (3) input the command(s) using the hand controllers until the arm is actually in the singularity (the caution light on the D&C panel comes on). The student must then (4) carefully attempt to drive the arm out of the singularity to attain a ghost arm goal position. (The timelimit is fairly tight for this portion of the task to ensure that the student is not using the software to resolve the singularity.)

Cameras and angles are prominently labeled for the two special views of the arm.

Advance Organizer:

"This task has several purposes: You will practice recognizing singularities BEFORE they are reached; you will plan how to avoid singularities; you will practice resolving singularities when necessary; and you will begin to learn about the important role of cameras in monitoring the arm.

"Each trial in this task requires four actions:

1. Identify the type of singularity that is most likely to occur (e.g., wrist yaw) given the initial configuration of the arm.
2. Determine the type of hand controller input required to drive the arm into the singularity (e.g., positive roll).
3. Actually drive the arm into the singularity.
4. Resolve the singularity.
5. Adjust the cameras appropriately throughout the task.

"Actions 1 and 2 are performed by using the mouse to select an answer from a set of possible answers. Actions 3 and 4 require manipulating the arm using the hand controllers. In Action 4, not only you must resolve the singularity, you also must drive the arm to a configuration indicated by a ghost arm so that you will be far away from the problematic arm configuration.

"As you perform these tasks, notice how camera selections and adjustments assist you in determining the arm position and attitude.

"This task is evaluated for correct responses (all actions) and efficiency (action #3). Efficiency in this context means unwasted movement, for example, not using the RHC when only the THC is required. In addition, efficiency may be measured by time."

HINT buttons: An editable number of buttons (currently three) are available that give definitions of singularities.

Hint text: "The shoulder yaw singularity occurs when the wrist yaw joint is within three feet of the shoulder yaw joint axis. The arm in this configuration experiences restricted motion that is normal to the pitch plane. Translation commands parallel to the Orbiter Y axis could be delayed as the shoulder yaw and wrist yaw joints are reconfigured. The arm swings about a considerable amount to achieve this and requires even keener vigilance by the operator to preclude contact.

"The wrist yaw singularity, sometimes called gimbal lock, occurs when the wrist yaw joint is within 15 degrees of +90 or -90. This creates a configuration where wrist pitch and wrist yaw are parallel and requires arm reconfiguration when roll about the pitch plane is commanded.

"The planer pitch joint singularity occurs when the three pitch joints are in a line, or more specifically, when the elbow pitch joint is at or above -7.6 degrees. Any degree of freedom that requires the arm to get longer is not

available since the arm is essentially at its maximum length. This situation is seen anytime the arm is at or near its cradled position."

**Levels of Complexity:**

- (1) Obvious singularities are used and good camera angles are given
- (2) Obvious singularities are used; student must adjust the cameras
- (3) Goal arms are more difficult to attain; singularities are less immanent, requiring the student to work harder to visualize and create them; cameras must be adjusted.

**Evaluation:** Number correct (on the first attempt). On the drive portion, make sure the student does not, for example, use the RHC when only the THC was required. When the timer is used, time allowed is 120 seconds to complete all phases of the task except for LOC 4, which gets 180 seconds. For the section on driving into singularities, accuracy is cut 50% if student flies into a reach limit; cut 75% if student flies to a soft stop; and becomes zero if student hits a hard stop.

**Advancement Criteria:** Student performs five consecutive trials without error. (Make sure that one of each type of singularity is presented within this set of trials without making it predictable which singularity will appear next.)

### 3.2 REACH LIMIT AVOIDANCE

When the arm is in a reach limit, it must be reconfigured by the arm operator before the RMS task can be resumed. In addition, the Orbiter should be in free drift to prevent the possibility of breaking the arm. Thus, the fewer reach limits, the safer and more efficient the operation.

Students initially learn about reach limits in core training (knowledge). To avoid reach limits, the student must know where they are encountered in order to recognize when the arm configuration is close to a reach limit. Positive and negative values of six arm positions (i.e., 12 actual positions) result in reach limits.

#### Advance Organizer

##### REACH LIMIT RECOGNITION AND AVOIDANCE

"A reach limit occurs when the arm is about to reach the end of travel for a particular joint. Positive and negative values of six arm positions (i.e., 12 actual positions) result in reach limits. (Refer to the Payload Deployment and Retrieval System Overview Workbook for details on reach limits.)

"When the arm is in a reach limit, it must be reconfigured by the arm operator before the RMS task can be resumed. In addition, the Orbiter should be in free



drift during reconfiguration to prevent the possibility of breaking the arm. Thus, the fewer reach limits you encounter, the safer and more efficient the operation."

### Task 3.2a: Recognizing Reach Limits

Because the P2T2 cannot display arms in different positions on the same screen, and because panel A8 indicates whether the arm is in a reach limit, three different screens are used for this task, each depicting a different arm configuration.

Three buttons are available for viewing the arm ("See Arm 1," "See Arm 2," "See Arm 3") and three buttons for selecting the arm that has a singularity ("Select Arm 1," "Select Arm 2," "Select Arm 3"). The student cannot use Spec 96 in this exercise except for wrist singularities.

#### Advance Organizer:

"A reach limit occurs when the arm is about to reach the end of travel for a particular joint. Positive and negative values of six arm positions (i.e., 12 actual positions) result in reach limits. (Refer to the Payload Deployment and Retrieval System Overview Workbook for details on reach limits.)

"When the arm is in a reach limit, it must be reconfigured by the arm operator before the RMS task can be resumed. In addition, the Orbiter should be in free drift before reconfiguration to prevent the possibility of breaking the arm. Thus, the fewer reach limits you encounter, the safer and more efficient the operation.

"The purpose of this task is to learn to recognize reach limits. Once you can do this reliably, you can then avoid reach limits altogether by careful planning.

"In this task, you will see three views of the same arm configuration in each window of a screen; there are three screens with different arm configurations on them—one of which is a reach limit. You must choose the arm that is in a reach limit.

"This task is evaluated on whether your choice is correct or not. A timer is used to limit the amount of time you have to view through the configuration and make a decision.

"Hint: There are six possible reach limits: Shoulder Yaw occurs at +/- 175.2 degrees; Shoulder Pitch occurs at +2.6 and +140.4 degrees; Elbow Pitch occurs at -2.4 and -155.6 degrees; Wrist Pitch occurs at +/- 114.4 degrees; Wrist Yaw occurs at +/- 114.4 degrees; and Wrist Roll occurs at +/- 440 degrees."

#### HINT buttons: An editable number of buttons (currently three) are available that give definitions of reach limits.

Hint text:

Reach Limits

Shoulder yaw	+175.4 and -175.4
Shoulder pitch	+2.6 and +140.4
Elbow pitch	-2.4 and -155.6
Wrist pitch	-114.4 and +114.4
Wrist yaw	-114.4 and +114.4
Wrist roll	-440.0 and +440.0°

Levels of Complexity:

1)-(6) Start with shoulder yaw; then shoulder pitch; then elbow pitch; then wrist pitch; then wrist yaw; then wrist roll. (Student masters each one separately.) For LOC 6 (wrist-roll reach limits), student can use Spec 96.

(7) Present the reach limit locations randomly

Evaluation: Accuracy is measured by number correct/incorrect. When the timer is used, task is allowed 30 seconds.

Advancement Criteria: For each reach limit type and LOC, the student must get three consecutive trials correct without error.

**Task 3.2b: Visualization Of and Driving the Arm Into Reach Limits**

In this task, the student sees an arm configuration which is almost in a reach limit. Panel A8 is displayed and one window is used for feedback; the other two windows show optimal views of the arm.

In this task, the student is given a series of arm configurations, one at a time. Panel A8 is used and one window is used for feedback. The other two windows show optimal views of a single arm configuration that is close to but not in a singularity.

In each trial, the student must identify (1) the location of the reach limit (shoulder yaw, shoulder pitch, elbow pitch, wrist pitch, wrist yaw, or wrist roll) that is most likely to occur from the given configuration, and (2) what THC and/or RHC inputs would result in that reach limit (e.g., positive roll). These activities are performed by using the mouse to move a cursor over textual selections in the feedback window and clicking the mouse button. A simple "Correct" or "Wrong" appears, and the student must keep trying (cannot proceed) until correct. Make sure student does NOT use Spec 96 during this exercise.

The student must then (3) input the command(s) using the hand controllers until the arm is actually in the reach limit (the REACH LIMIT annunciator on the D&C panel comes on). Make sure they have the D&C panel up if possible, and tell

them to do so before the task starts if they don't. (4) Student must resolve the reach limit without encountering a software stop.

Cameras and angles are prominently labeled for the two special views of the arm.

**Advance Organizer:**

"This task has several purposes: You will practice recognizing reach limits BEFORE they occur; and you will plan how to avoid reach limits; and you will learn more about the important role of cameras in monitoring the arm.

Each exercise in this task requires four actions:

1. Identify the location of the reach limit that is most immanent (e.g., + wrist pitch) given the configuration of the arm.
2. Determine the type of hand controller input that would drive the arm into the reach limit (e.g., positive roll).
3. Actually drive the arm into the reach limit until the reach limit annunciator on the D&C panel is activated.
4. Resolve the reach limit without encountering a software stop.

"Actions 1 and 2 are performed by using a mouse to select the answer from a set of possible answers. Actions 3 and 4 require manipulating the arm using the hand-controllers.

"As you perform these tasks, notice how the camera selections and adjustments assist you in determining the arm position and attitude.

"This task is evaluated for correct responses (all actions) and efficiency (actions #3 and #4). Efficiency in this context means unwasted movement, for example, not using the RHC when only the THC is required. In addition, efficiency may be measured by time."

**HINT buttons:** An editable number of buttons (currently three) defining reach limits are available; these disappear after use (i.e., student performs any action).

Hint text:

**"Reach Limits**

Shoulder yaw	+175.4 and -175.4
Shoulder pitch	+2.6 and +140.4
Elbow pitch	-2.4 and -155.6
Wrist pitch	-114.4 and +114.4
Wrist yaw	-114.4 and +114.4
Wrist roll	-440.0 and +440.0

"In order to drive the arm to the low elbow pitch reach limit, you must pass the planer pitch singularity, and continue driving the arm until the reach limit warning light is turned on."

**Levels of Complexity:**

- (1) Obvious singularities are used and good camera angles are given
- (2) Obvious singularities are used; student must adjust the cameras
- (3) Goal arms are more difficult to attain; singularities are less immanent, requiring the student to work harder to visualize and create them; cameras must be adjusted.

Evaluation: Accuracy = Number correct (on the first attempt). On the drive portion, the student must (1) immediately begin the correct input(s) to the HCs (i.e., not start using the THC when only the RHC is required), and (2) drive the arm until the REACH LIMIT annunciator light comes on. Regarding camera use from LOC 2 on, student must at least select the most appropriate camera. Efficiency = driving the arm out of the reach limit and time when used.

Also: If the student gets into a software stop, he/she loses 50% (editable) of accuracy score. If the student hits a hard stop, this results in losing 75% (editable) of accuracy score.

Advancement Criteria: Student performs three consecutive trials without error, except for complexity #2, where one of each type should be included, and therefore the student must perform six consecutive trials without error.

**Task 3.2c: Avoiding Reach Limits; Beginning Camera Skills**

The student sees panel A8 and two views of the arm; the fourth window is used for the performance indicator and feedback. The arm configuration shown is close to a reach limit for one of the joints, and the student must drive the arm to a new position, indicated by a ghost arm target, while avoiding the reach limit. (If

the student hits a reach limit, the trial is over and unsuccessful; however, the next trial should use the exact same initial configuration.)

Cameras and angles are prominently labeled for the two special views of the arm.

**Advance Organizer:**

"The two purposes of this task are to (1) avoid driving the arm into reach limits when performing a fly-to, and (2) use intelligent camera selection and adjustment to help you succeed.

"You may remember that, when the arm is in a reach limit, it must be reconfigured by the arm operator before the RMS task can be resumed. In addition, the Orbiter should be in free drift during reconfiguration to prevent the possibility of breaking the arm. Thus, operators should consciously plan to avoid reach limits before moving the arm.

"In this task, the arm is configured close to a reach limit. Your job is to fly the arm until it is superimposed over the ghost arm target, but without driving into the reach limit. This requires careful planning on your part!

"Also in this task, you must make camera selections and adjustments yourself. The cameras can help you understand the arm position and attitude better.

"This task is evaluated for efficiency, safety, and camera selection. Efficiency is measured by how directly you fly the arm to the target attitude (without hitting a reach limit), and of time (when used). Safety is measured by avoidance of reach limits. Camera usage is based, minimally, on selecting the most appropriate camera(s).

"Set up your camera(s) and make your plans before you move the arm. Once you move the arm (if the timer is used) you have 60 seconds to attain the ghost arm position."

**Levels of Complexity:**

- (1) Have student master one reach limit at a time, with the new coordinates easy to attain while avoiding the reach limit; make sure positive and negative values of the reach limits are included per mastery sequence.
- (2) As for (1), but with the type of reach limit presented randomly; use "vanilla" cameras and angles
- (3) Make the new coordinates difficult to attain without driving into the obvious reach limit

- (4) Make the new coordinates difficult to attain without driving into either the obvious reach limit or especially one that is NOT the most obvious reach limit

Evaluation: (1) Efficiency is measured by how directly the student flies to the target position without hitting a reach limit. Also, when time is used, 1/3 of the efficiency measure is based on time: by the seconds required to reach the new coordinates. (Start when the student moves HCs.) (2) Safety is measured by avoidance of reach limits. (3) Time allowed is 60 seconds from the first movement of the arm when the timer is used. (4) Camera use: Student should at least select the most appropriate camera(s).

Advancement Criteria: Student must perform three consecutive trials at each level of complexity without error. However, for complexity (1), the student must exhibit mastery with six different reach limits (at least 18 trials to advance).

### 3.4 CAMERA SKILLS (BEST VISUAL INFORMATION)

Since camera usage is task specific (i.e., the best information is dependent upon which RMS task is being performed and where the arm is located), the student will not have had experience with some RMS tasks and related cameras/angles. For example, although the selection of camera perspectives is always important, visual information becomes increasingly more important the closer the arm is to other structures. Successful grappling, ungrappling, berthing, and unberthing are highly dependent upon the operator selecting a perspective that helps him or her see the results of hand-controller inputs and the relationship of the arm structure to other structures.

One problem with teaching and evaluating camera skills is that individuals differ on what view makes sense to them. The main point in these tasks is to ensure that the student is looking at the critical points of interest from a reasonable distance at the proper time in a procedure. For example, when unberthing, students must first view the trunnion/v-guide relationship very closely; and shortly after starting to unberth, must find a broader perspective to monitor payload and Orbiter structures for collision. In such a case, whether camera C or D is chosen for the broader perspective might be irrelevant, frequently depending on the payload characteristics or simply what makes sense to the operator.

#### Advance Organizer

#### CAMERA SKILLS

\*Camera usage is task specific; that is, the best information is dependent upon which RMS task is being performed and where the arm is located.

\*Although the selection of camera perspectives is always important, visual information becomes increasingly critical the closer the arm is to other structures. Successful grappling, ungrappling, berthing, and unberthing are highly dependent on selecting a

perspective that helps you see the results of hand-controller inputs and the relationship of the arm structure to other structures.

"Individuals differ on what view gives them the best information. For this reason, the main point in the following tasks is to ensure that you are looking at the critical points of interest from a reasonable distance at the proper time in a procedure. For example, when unberthing, you must first view the trunnion/v-guide relationship very closely; and shortly after starting to unberth, must find a broader perspective to monitor payload and Orbiter structures for collision. In such a case, whether camera C or D is chosen to get the broader perspective might be irrelevant, depending on the payload characteristics or simply what makes sense to the you at the time."

#### **Task 3.4a: Demonstration: Using Cameras for Specific Tasks**

This demonstration uses the normal P2T2 screen. Since animation is not possible without modifying the P2T2, this demonstration uses a series of "stills." Each still shows camera use at a significant point in the deploy/retrieval process. A "CONTINUE" button is used for the student to progress through the demonstration.

The main point in adjusting the camera is to understand what you're looking at. For this reason, selecting a view with the camera is somewhat individualistic. We try here to show one idea of what good use of the cameras is, and to point out why each view is selected.

The student sees views of the arm in various positions as it moves from pre-cradle position to actually grappling the Spartan payload (which has a grapple fixture on the opposite side of the bay from the arm). Cameras/angles are demonstrated as follows:

##### **Deploy:**

- **At pre-cradle**

Cameras should start as they would be after uncradling: Camera B (or C) would be looking at the aft or mid MPM; camera A at the forward MPM

- **As the arm moves from pre-cradle towards the payload**

One monitor would use the EE camera. In another monitor, camera C or D would point towards the grapple fixture of the payload (if visible), and would view the EE from 4-6 feet away.

– Pre-grapple position and grappling

Student should be viewing the EE from about 4-6 feet away; both the grapple fixture and the EE should be on the screen simultaneously and viewed from the side so the approach of the EE to the grapple fixture can be monitored and the distance before grapple gauged correctly. EE camera is still in use. When the student can see the grapple fixture in the EE camera, mode should change to EE mode.

– Unberthing

In one camera, student should see starboard V-guides; in the other camera, port V-guides. These cameras should be zoomed in fairly close to monitor contact between trunnions and V-guides.

– As trunnions leave the V-guides (i.e., about 3 inches up from the bottom of the guides)

Zoom cameras out and pan to see the side of the payload and side of the payload bay to make sure no contact is made between structures.

– From unberthed to a mid-point in the deploy procedure

Take the arm to a point where a reach limit or singularity is inevitable without an arm reconfiguration. Move the cameras in to see the problem area and reconfigure the arm.

– Fly-to deploy

Student should use alternate views to view both the payload and its position relative to the bay. For example, if one camera shows a port view, the other should be starboard; or, if one points aft, the other should point fore. (If not, the visual perspective of the attitude and position of the arm/payload in the bay can be misleading.)

– Deploy

When releasing the payload, have a side view that shows all of the EE, wrist joints, and payload around the grapple fixture. Other camera shows EE view.

When done, the student can select to see the demonstration over or go on to the next demo.

---



A similar follow-on demonstration is also given of retrieving the payload: the demonstration shows cameras/angles as follows:

- Fly-to with reach limit/singularity bypass  
  
Arm must fly from pre-cradle to retrieval position. Move the camera back after the arm is reconfigured to watch the arm relative to structures while the arm moves to pre-grapple.
- As the arm moves from pre-cradle towards the payload  
  
One monitor would use the EE camera. In another monitor, camera C or D would point towards the grapple fixture of the payload (if visible), and would view the EE from 4-6 feet away.
- Capture  
  
(Incomplete)
- Fly payload towards payload bay  
  
Two cameras view the same area from different sides (at 90 degrees from each other), where the goal is to give the student a good view of the arm's position in space. Occasionally, the camera view could change briefly to show one of the joints, the POR, or even the goal POR.
- Berth payload and return arm to pre-cradle  
  
One camera should point to a port V-guide and the other should point to a starboard V-guide. Also, zoom into both V-guides so they appear to be the same size: There is a mark on the V-guide that all the trunnions should pass at the same time. If they don't, then the payload is not lined up correctly. (This won't happen in P2T2 for Unberth because P2T2 always works precisely, unlike the real RMS, which is analog and doesn't always move in exactly the commanded direction). Also, this view should only be in effect until the trunnions clear the V-guides. After that, general camera views take effect.

When done, the student can select either demo again or go on to the next task.

Advance Organizer:

"The purpose of this demonstration is to acquaint you with optimal camera strategies during the various RMS operations. Although exact camera usage is different from operator to operator, this demo will give you some idea of the possibilities.

"This demonstration will take you from pre-cradle to pre-grapple, through the grapple sequence, unberthing, flying to the deploy position, and

deploying. The opposite sequence (retrieving and berthing) is the same process, only in reverse, and is not explicitly modeled.

"Your job here is to understand how each camera view was created and why it is useful for the task at hand. What are the critical features of the task that you need to see? Which cameras are used? How are the cameras adjusted? Is there some aspect of the task that you cannot see and would therefore adjust the cameras differently?

"This demonstration is self-paced. When you have studied it, you can review it if desired. You are not evaluated."

- . Performance Indicator: NA
- . Levels of Complexity: NA
- . Evaluation: NA
- . Advancement Criteria: Student must watch both demonstrations.

#### **Task 3.4b: Finding Optimal Camera Views**

Like Task 3.4a, this exercise uses a series of "stills." One window contains panel A8, one has the performance indicator, one window contains a pre-designed view of the RMS task, and one view is adjusted by the student.

In this exercise, the same two series of stills are given as in task 3.4a, only a larger payload is selected (which may have the grapple fixture facing the arm). However, this time for each still, one of the windows already shows an excellent view (camera, angle, and range) for the impending subtask. The student must then achieve the same specific camera perspective in the second window. If possible, the given view is not even labeled per camera (e.g., "Camera C" is not displayed). Student gets a DONE button as well as the timer (when the timer is used).

For LOC 1, the empty window is labeled "DUPLICATE THE VIEW TO THE LEFT."  
For LOC 2, the empty window is labeled "FIND A DIFFERENT, USEFUL VIEW FOR MONITORING THIS TASK."

- . Advance Organizer:

"The purpose of this task is to help you learn which camera(s) and camera views are most useful for the various RMS tasks, and to find these cameras and views yourself.

"There are two sections to this task. In the first section, you will see a window with the camera adjusted in an optimal way for some particular RMS task. Your job is to use the other window to select the same camera and adjust it to the same view as given. Because only one camera is used

per operation, there may be information missing that you would want if you were actually performing the job; in the next section of this task, you will have a chance to correct this problem.

"In the second section of this task, you will see one view of the RMS operation and must create another useful view in a different window.

"This task is evaluated by correct or incorrect solutions. After you have created a view, press the DONE button; or, your solution will be evaluated after 60 seconds (if the timer is running). However, if you are not within a certain range of acceptable performance, you will be asked to continue working on the problem. In this way, you will be sure to understand the available cameras and their applicability during RMS operations."

HINT buttons: Five Camera Hint buttons are available per phase (i.e., deploy, retrieval) that are context-specific; when the student uses these, they disappear. Text tells which camera was used for the optimal view and how the camera was panned or zoomed to achieve the view.

Levels of Complexity:

- (1) As above
- (2) Run the same stills again. However, have the student adjust the other camera to give another (different) useful view.

Evaluation: Correct/incorrect camera and: for LOC 1, within approximately 15% angle and range of the given view; for LOC 2, student must simply have the targeted object or view on the screen, e.g., during grapple, if the given view is EE, then the student must have the grapple fixture and EE (when close enough to the grapple fixture) in view with another camera. Student has 60 seconds to complete the task when the timer is used.

Advancement Criteria: Student must complete the entire exercise. (Student must continue to work with the cameras until correct.)

## Part Tasks for Procedural Skills

### 4.0 MODE SELECTION

#### 4.1 REACH LIMIT RECOVERY

Students have learned in other tasks (see the Monitoring section) how to recognize reach limits, how to avoid them, how to drive into software stops from reach limits, but not how to recover from reach limits and software stops. Earlier, optimal camera views for seeing and resolving reach limits were always supplied to the student.

LOCs: Except in a few places, a minimum of three levels of complexity (LOCs) will be implemented per task.

#### Task 4.1a: Exercise: Use of SINGLE Mode; Developing Camera Skills

Single-joint drive control mode—or simply, single mode—is explained in the advance organizer: what it is, how it's selected on the P2T2, and its use in resolving software stops (including a reminder of how SPEC 96 highlights the problem joint). A description of the task is given, and a reminder that cameras can be changed to get a better look at a particular joint configuration or movement is included.

The P2T2 is configured so that only one large window is used. The student uses the arm to select each joint and drive it slowly in both positive and negative directions in SINGLE mode (no other mode is allowed).

This exercise is simply monitored to ensure that the student (1) tries each joint, (2) moves the arm in positive and negative directions per joint, (3) tries the various types of movement available for the joint (such as roll, pitch, yaw. Twelve combinations of joints/movement/positive/negative are possible and should be explored.), and (4) makes some use of the cameras (selects the correct camera, adjusts it within approximately 25% of an optimal setting).

#### Advance Organizer:

"The purpose of this exercise is to practice using single-joint drive control mode—or simply, single mode. Among other uses, single mode is used to resolve reach limits.

"Single mode allows you to move the arm on a joint-by-joint basis with full GPC support. You will have full use of joint drive characteristics on a joint-by-joint basis. Single mode as well as the particular joint you want to drive are selected using the relevant rotary switches on panel A8U (operated by the knobs you see on the side of the P2T2).

"In this exercise, you will select the various joints and practice driving them in positive and negative directions. Don't forget to try out the rotational movements as well as the translational. Also, be sure to change the cameras to get optimal views of the joint you are driving.

"This exercise is self-paced."

- . HINT buttons: Using SPEC 96.
- . Levels of Complexity: NA
- . Evaluation: NA
- . Advancement Criteria: Student presses the NEXT TASK button when finished. However, encourage the student (as described under "Coaching" above) to continue this exercise until all joints/movements have been explored.

#### **Task 4.1b: Resolving Reach Limits; Developing Camera Skills**

The P2T2 is configured so that one window shows panel A8 and one window is used for the performance indicator. In the other two P2T2 windows, two views show the arm already in a reach limit. Both of these windows, which use different cameras, have a "vanilla" view of the arm: that is, they use randomly-chosen cameras and default angles/positions. The student must use the hand controllers to resolve the reach limit. If the arm hits a software stop, the student has failed the trial and the trial must be reset at the beginning.

In this task, camera skills are also evaluated. The student must use the most appropriate camera for the reach-limit joint, and adjust the camera within an acceptable range for viewing the problem.

- . Advance Organizer:

"The purpose of this task is to increase your awareness of reach limits, software stops, and the importance of camera usage. You'll also get to solve some challenging problems for performing fly-to's while avoiding reach limits.

"In the first section of this task, the arm starts in a reach limit and you simply resolve the reach limit without driving the arm into a software stop. When the D&C Panel reach limit annunciator light goes out, you are done.

"In the other sections of this task, a ghost arm target is provided, and you must fly the real arm until it is superimposed over the ghost arm, again without driving it into a reach limit. This time, if you do hit a reach limit, you must resolve the reach limit without incurring a software stop. Software stops are "fatal errors" in this task!

"In both sections of this task, you must select and adjust at least one camera to a reasonably optimal view for the task.

"This task is evaluated for accuracy, efficiency, safety, and camera usage. Accuracy is measured by resolving reach limits without encountering any software stops. Safety means not encountering software stops. Camera usage is based on selecting at least one camera and adjusting it well. Efficiency is a function of time (when the timer is used).

HINT buttons: Five (editable) camera hint buttons are available that give context-sensitive information on which camera is best and on how to adjust the camera. These buttons disappear after use.

Levels of Complexity:

- (1) Student must simply move the arm out of the reach limit without encountering a software stop. This is signified by the silencing of the reach-limit alarm and by the annunciator light turning off.
- (2) The arm is in a reach limit. A ghost arm is used as the target arm position. Achieving the ghost arm position is difficult without encountering a software stop.
- (3) The arm is in one reach limit, and another joint is close to being in a reach limit; moving out of the one without incurring the other is difficult. A ghost arm is used as a target position. In this case, the student must merely resolve the second reach limit (if encountered) as well, again without creating a software stop. Hitting a reach limit is not counted as an error in this trial, but hitting a software stop is a fatal mistake.

Evaluation: LOC 1: Student resolves any reach limits without encountering a software stop. LOC 2 and 3: Student achieves ghost arm position without encountering a software stop. When the timer is used, time limit for LOC 1 is 30 seconds; for LOC 2, three minutes; and for LOC 3, five minutes.

All LOCs: Correct/incorrect selection of camera and within some range (e.g. 25%) of optimum angle/position; note that the student need only configure one camera correctly, regardless of which window he/she uses.

All LOCs: Camera usage: based on percent of time at least one camera has an acceptable view of the arm.

Advancement Criteria: LOC 1: Student can perform three trials consecutively without error. LOC 2: Student can perform three trials consecutively without error. LOC 3: Student can perform four trials consecutively without error; or can perform three trials consecutively without error and without encountering another reach limit.

## 4.2 SOFTWARE STOPS

If an arm operator is in a reach limit and inputs inappropriate commands to the hand controllers, the arm will hit a software stop. Because of this first-then relationship, software stops should be taught in conjunction with reach limits. At the point where a software stop is encountered, the arm can only be driven one joint at a time using SINGLE mode. Obviously, one way to avoid software stops is to avoid reach limits. However, once the arm is in a reach limit, the student needs to know the kinds of arm movement that will cause a software stop and drive the arm in some other direction.

### Advance Organizer

#### SOFTWARE STOPS

"If you drive the arm into a reach limit and do not then drive it out properly, you will encounter a software stop. Software stops bring the arm to rest before the absolute travel limit, the mechanism stop (hard stop) is reached. (Refer to the Payload Deployment and Retrieval System Overview Workbook for more information on software stops.)

"Once the arm is in a software stop, you must use single, direct, or backup mode to drive it out of the soft stop limit. If, in single mode, you again drive the arm into a soft stop, the arm is again brought to rest, and you must re-enter single mode and try again.

"You can see that avoiding reach limits is the best policy. However, you will also want the skill of recovering successfully from a software stop."

### Task 4.2a: Driving Into and Resolving Software Stops

P2T2 screen is as for task 3.2c. The arm is shown in a reach limit. The student must first drive the arm into a software stop; this indicates the student's understanding of the kinds of arm movement (and prior inputs) that will cause a software stop from the particular reach limit. The student must then resolve the software stop using single mode. Make sure the student is viewing the D&C panel, or tell the student to call this panel up before proceeding.

Cameras and angles are prominently labeled for the two special views of the arm.

#### Advance Organizer:

"The purpose of this task is to give you practice in driving the arm out of reach limits. As you perform this task, note how camera selections and adjustments help you understand the problem and its solution.

"This task has two parts: You will first see the arm in a reach limit. Your job is to drive the arm INTO the reach limit until you hit the software stop. Be aware of what movements you input to do this.

"P2T2 will then reset to the original state (i.e., the arm will again be in the reach limit). In the second part, your job is to drive the arm OUT of the reach limit without hitting the software stop. In this case, you must use single mode to resolve the reach limit. Note that the task is "complete" when the reach limit annunciator light goes out.

"This task is evaluated for accuracy (correct/incorrect). In addition, efficiency is a function of how directly you fly the arm to the target position (e.g., software stop), and of time (when used).

HINT buttons: Five buttons are available (an editable number) that give angles for reach limits and software stops; a similar table can be found on page 2-23 of the PDRS OV 2102 Workbook. These buttons disappear after use.

Levels of Complexity:

- (1) Student masters one reach limit/software stop at a time
- (2) Reach limit/software stops are presented randomly

Evaluation: Correct/incorrect. Efficiency: Student should move the arm within 115% of the shortest possible path. When time is used, allow 30 seconds for phase 1, and 60 seconds for phase 2 of the task.

Advancement Criteria: Student performs three consecutive trials (each trial has 2 parts) per reach limit (LOC 1) or complexity level (LOC 2) without error.

#### **Task 4.2b: Avoiding Software Stops; Beginning Camera Skills**

This task is very much like 4.2a, except it is more of a "test" than an exercise. In this task, the arm is again placed in a reach limit and the student must drive the arm out of the reach limit without driving it into a software stop.

Since this task is really a continuation of skills learned in 3.2c and 4.2a, avoiding reach limits and software stops should be a familiar task by now. In addition, student awareness of camera/angle possibilities and importance should have been heightened in the previous exercises by virtue of the prominent labels on the windows and the usefulness of a good view of the appropriate arm joints when trying to locate and/or avoid singularities, reach limits, and software stops. For this reason, we require the student to deliberately find a good camera angle for one of the two windows. Camera and angle is prominently labeled for the one given view of the arm. Thus, panel A8, a window for the performance indicator and feedback, one view of the arm that is not the best possible view, and a blank window labeled "USE THIS WINDOW TO FIND THE BEST CAMERA VIEW AND ANGLE FOR THIS TASK" are used. Task begins with no mode selected.



Advance Organizer:

"The purposes of this task are to (1) develop your skills in resolving reach limits, and (2) improve your skills in selecting and adjusting the camera(s) for the optimum view of the arm.

"In this task, the arm starts in a reach limit. You simply drive the arm out of the reach limit without driving into a software stop. Of course, driving into a different reach limit is not acceptable!

"This task is evaluated for accuracy, efficiency, and camera usage. Accuracy is a correct/incorrect measurement (i.e., you drive the arm out of the reach limit without hitting a software stop or another reach limit). Efficiency is a function of how directly you fly the arm, and of time (when used). And camera usage involves selecting the best camera and camera angles.

"Note that the task is "complete" when the reach limit annunciator light goes out."

HINT buttons: Five Camera Use buttons (an editable number) should be available that disappear after use. These hints are context-specific and tell which camera gives an optimal view of the arm, and how to adjust the camera.

Levels of Complexity:

- (1) Student masters one reach limit at a time
- (2) Reach limits are presented randomly
- (3) A different reach limit is "near by"

Evaluation: Correct/incorrect, meaning student drives the arm out of the reach limit without hitting a software stop; camera correct/incorrect (exact camera; zoom/pan should be within some range) by the time the student finishes the task (i.e., reach limit light goes out). Efficiency: Student should move the arm within 115% of the shortest possible path. When the timer is used, time allowed is 60 seconds from when student moves the arm.

Advancement Criteria: Student performs three consecutive trials in each complexity level without error

### 4.3 GRAPPLING

Grappling is a task that involves moving the EE from the pre-grapple position-and-attitude to a position and attitude that permits rigidization. Grappling from the pre-grapple position is a relatively simple task; the more difficult aspect of grappling is preparing to grapple (i.e., placing the arm in the pre-grapple position and attitude). Safety, efficiency, and accuracy have special meanings when grappling.

#### Task 4.3a: Grapple Procedure; Camera Skills

The P2T2 is configured with two windows showing the arm, another window with panel A8, and one window for the performance indicator and feedback. The arm is placed in the pre-grapple position; software should be in Vernier rate. Cameras are at optimal angles/positions. In addition, a visual aid consisting of a cross hairs is used as though it were taped to the CRT (NOT as though it were part of the wrist camera); this visual aid is of the exact length that, when the EE is close enough to the target for rigidization, the end bars line up with existing target indicators on the payload.

The student must grapple the payload. He/she is reminded to monitor the talkbacks on panel A8 when grappling. When the student presses the grapple trigger and the rigidization talkback has changed to gray, the student must turn EE Auto mode OFF. The trial is complete when all talkbacks have turned to grey.

#### Advance Organizer:

"Grappling is a task that involves moving the End Effector from the pre-grapple position-and-attitude to a position-and-attitude that permits rigidization. The purpose of this task is to give you initial practice in grappling a payload. The "hard part" of grappling has been done for you: the arm starts in pre-grapple position, lined up and ready for the grapple procedure.

"To accomplish this task, you will use the Grapple procedure in the Procedures section of the Flight Data File. Also in this task, except for the first level of complexity, you must select and adjust both camera views: one from the wrist camera, viewing the target, and one viewing both the end effector and grapple fixture from the side.

"In addition, a visual aid consisting of a cross hairs is used in the wrist camera view as though it were taped to the CRT (NOT as though it were part of the wrist camera); this visual aid is one of the exact length that, when the end effector is close enough to the target for rigidization, the end bars line up with existing target indicators on the payload.

"During grapple, be sure to monitor the talkbacks on panel A8. When all three talkbacks have changed from barberpole to grey and you have turned EE Mode Auto to OFF, and the Payload IDs are set correctly, you have completed the grapple procedure.

"This task is evaluated for safety, accuracy, efficiency, procedure and camera usage. SAFETY dictates that no part of the arm come in contact with any other structure; in addition, you must use Vernier rate while you are close to the orbiter. ACCURACY is a measure of tolerance for the EE and the grapple fixture, both regarding position and attitude. EFFICIENCY is a function of time, and of path between the grapple position and pregrapple position. PROCEDURE evaluation examines whether you have followed the correct sequence of actions in ungrapppling/releasing the payload. CAMERA usage is measured by the percentage of time you used the cameras correctly.

"The higher levels of complexity take away some of the earlier cues, so visual and operational adjustments may need to be made. Time varies per level of complexity and begins when you move the hand controllers. When you have finished the task, use the mouse cursor to press the "Done" button."

HINT buttons: After LOC 1, an editable number of buttons (currently five) are available that each bring the visual aid back for 30 seconds.

Levels of Complexity:

- (1) As stated above
- (2) Start cameras at "vanilla" positions
- (3) Use different positions for grapple fixtures (top, port, starboard). Student must also select Vernier rate in this LOC.

Evaluation:

Correct procedural performance: Student correctly performs steps of the procedure (i.e., move into correct position, "rigid" talkback ON, "close" talkback ON, "capture" talkback ON, presses grapple trigger) in the correct order. In addition, when timer is used: LOC 1: completed within 60 seconds; LOC 2-4: completed within 3 minutes

Grappling performance is acceptable as follows:

- A "grapple envelope" is defined for acceptable tolerances between the EE and the grapple fixture for when student begins the grapple procedure: (1) an area of plus or minus 4 inches in Y or Z of the grapple fixture; (2) plus or minus 15 degrees error in pitch or yaw and plus or minus 10 degrees error in roll
- External edge of the EE can be flush with the grapple fixture plate or within 3.9 inches from the plate.

**Efficiency:** Student should move the arm within 115% of the shortest possible path. Furthermore, student should use both hand controllers simultaneously 75% of the time, where the student improves the arm attitude at the same time that they're moving the target towards the center of the CCTV.

**Safety is defined as:** No contact between the arm and anything else; also, arm must be in Vernier rate when within 5 feet of the payload.

**Camera skills for LOC 2 and higher:** Correct/incorrect selection of camera and within some range (e.g. 25%) of optimum angle/position; note that the student need only configure one camera correctly, regardless of which window he/she uses.

**Advancement Criteria:** Student performs three trials consecutively without error and within time limits per LOC.

#### **4.4 UNGRAPPLING**

Ungrappling is a task that includes the derigidization process and moving the EE from the grappled position-and-attitude to the pre-grapple position-and-attitude. Ungrappling is a simple task. Safety, efficiency, and accuracy have special meanings when ungrappling.

##### **Task 4.4a: Ungrapple Procedure; Camera Skills**

The P2T2 is configured with two windows showing the arm, one window with panel A8, and one window for the performance indicator and feedback. A grappled payload is displayed in a "deploy" position. Cameras are at optimal angles/positions. The student must ungrapple the payload and back the arm off to pre-grapple position; this position is indicated by a ghost arm during LOC 1. The student is reminded to monitor the talkbacks on panel A8 when ungrappling. The trial is complete when the three derigidization talkbacks have changed to gray.

##### **Advance Organizer:**

"Ungrappling is a task that includes the derigidization process and moving the end effector from the grappled position-and-attitude to the pre-grapple position-and-attitude. The purpose of this task is to give you experience in ungrappling a payload.

"In this task, you will be using both the Ungrapple and the Release procedures in the Procedures section of the Flight Data File. The two procedures are essentially the same, except for the setting of the Payload ID. In the first level of complexity, you will be using the Release procedure; in the second, you will be using the Ungrapple procedure; in the last, the procedures will be alternated. Also in this task, except for the first level of complexity, you must select and adjust the camera views: one should be

the wrist camera viewing the target, and the other should view both the end effector and the grapple fixture from the side.

"In addition, a visual aid consisting of a cross hairs is used in the wrist camera view as though it were taped to the CRT (NOT as though it were part of the wrist camera). This visual aid is of the exact length that, when the end effector is close enough to the target for rigidization, the end bars line up with existing target indicators on the payload.

"During ungrapple, be sure to monitor the talkbacks on panel A8. When the DERID and OPEN talkbacks have changed from barberpole to grey, be certain that you slowly pull the end effector away from the payload to a distance where you can see the grapple pin. Wait. When all three talkbacks have changed to grey turn the EE mode Auto to Off. Proceed to the pre-grapple position-and-attitude, which is indicated by a wire-arm. When you put the RMS BRAKES ON (and set the Payload IDs in the Release procedure), then you have completed the trial.

"This task is evaluated for safety, accuracy, efficiency, procedure and camera usage. SAFETY dictates that no part of the arm come in contact with any other structure; in addition, you must use Vernier rate while you are close to the orbiter. ACCURACY is measured by the final proximity of the real arm to pregrapple position after ungrappling has been finished. EFFICIENCY is a function of time, and of the path between grapple position and pregrapple position. The PROCEDURE evaluation examines whether you have followed the correct sequence of actions in ungrappling/releasing the payload. CAMERA usage is measured by the percentage of time you used the cameras correctly.

"The higher levels of complexity take away some of the earlier cues, so visual and operational adjustments may need to be made. Time varies per level of complexity and begins when you move the hand controllers. When you have finished the task, use the mouse cursor to press the "Done" button."

**Levels of Complexity:**

- (1) As stated above
- (2) Start cameras at "vanilla" positions
- (3) Use different positions for the grapple fixture (top, port, starboard)

Evaluation:

Correct procedural performance: Student correctly performs steps of the procedure (i.e., move into correct position, "rigid" talkback ON, "close" talkback ON, "capture" talkback ON, presses <return> key) in the correct order. In addition, when the timer is used, LOC 1: Completed within 60 seconds; LOC 2-3: Completed within 2 minutes

Efficiency: Student should move the arm within 115% of the shortest possible path.

Safety is defined as: No contact between the arm and anything else; also, arm must be in Vernier rate when within 5 feet of the payload.

Camera skills for LOC 2 and higher: As for task 4.3a

Advancement Criteria: Student performs three trials consecutively without error and within time limits per LOC.

#### 4.5 BERTHING

Berthing is a complex task. It requires a combination of skills, which include skill in using the hand controllers simultaneously, camera use skills, and visualization skills. It also requires a high degree of accuracy and a constant concern for safety.

##### Task 4.5a: Berthing a Payload; Camera Skills

A grappled payload is positioned directly over the V-guides in low hover position ( $Z = -650$ ). This is a small-sized payload with the grapple fixture on the top. The student must berth the payload and press the DONE button when finished. Visual aid: Lines are graphically displayed on the screen which are perpendicular to a line through the midpoint of the bottom of the V-guides and tangent to the bottom of the V-guides.

In this task, camera skills are also evaluated in LOC 2 and higher. The student must use the most appropriate camera(s) and adjust the camera(s) within an acceptable range for berthing the payload.

Advance Organizer:

"The purpose of this task is to help you develop your ability to BERTH a payload. Berthing is a complex task. It requires a combination of skills, which include skill in simultaneous use of the hand controllers, camera usage, and visualization. It also requires a high degree of accuracy and a constant concern for safety.

"In this task, the payload starts out positioned directly over the V-Guides ("low hover"). At first the cameras are set up in optimal positions, although

you can change them if you wish. Very quickly, you'll see that you must actually select and adjust them yourself.

"As you perform this task, remember that the P2T2 allows structures to "pass through" one another visually; nevertheless, proper performance does not allow this! If the payload trunnions pass through the V-guides, you have probably "damaged" either the payload, the Orbiter, or both.

"This task is evaluated for safety, accuracy, efficiency, procedure, and camera usage. Safety dictates that no part of the arm or payload come in contact with any other structure; in addition, you must use Vernier rate within ten feet of the payload. Accuracy is a measure of tolerance for the clearance between the trunnions and the V-guides. In normal RMS usage, you are allowed to "scrape" the V-guides with the trunnions; however, in this exercise, you are not. Efficiency is a function of how directly you move the arm/payload to the target position, using both hand-controllers simultaneously, and of time (when used).

"This task is divided into three levels of complexity. The first level of complexity gives you a ghost guide bar to follow and the cameras automatically reset for you. Be aware of the views given so you can recreate them. After the first level of complexity, camera skills are evaluated by your selection and adjustment. The higher levels of complexity will ask you to adjust one or both of the camera views, and to set up the RMS for berthing. Time varies per level of complexity and begins when you move the hand controllers.

"When you are satisfied that you have successfully finished the task, use the mouse cursor to press the "Done" button."

**HINT buttons: Gods-eye View Buttons:** These buttons would give a gods-eye view of the shuttle for 60 seconds at a time each. Each time a button is used, it disappears. Student starts with eight buttons (an editable number). [Have the initial quantity of buttons editable from the instructor editing file.]

**Line Aid Buttons:** Like the Gods-eye View buttons, these buttons would allow the student to return the visual aid lines for 60 seconds each. Student starts with 5 buttons (an editable number), which don't appear until LOC 2. [Initial quantity also needs to be editable by instructors.]

**Levels of Complexity:**

- (1) As above
- (2) Remove the visual aid and start the cameras in "vanilla" position
- (3) Change the position of the grapple fixture (top, port, starboard positions). (Unless LOC 2 is implemented, remove visual aids

except godseye at this LOC and start with cameras in "vanilla" position.)

- (4) Have the arm start farther away and more and more off-center (i.e., not straight up from the payload).
- (5) Use a medium and then a large payload

**Evaluation:**

Performance is safe and accurate as long as the arm and payload do not make contact with anything (in the MDF, the trunnions and V-guides can touch, but not hard; however, this is impossible to measure in the P2T2) and the student is using Vernier rate whenever moving within 5 feet of the payload.

Efficiency: Student should move the arm within 115% of the shortest possible path. Hand controllers should be used simultaneously 50% of the time.

Accuracy: Accuracy is based on the student getting the payload into the payload bay.

Safety: Every time the student hits the V-Guides with the trunnions, he/she gets -15 points from possible 100. Student must get 75 points or more to pass safety.

Time limits when used: LOC 1: 3 minutes; LOC 2 and 3: 5 minutes; LOC 4: 7 minutes; LOC 5: 7 minutes for medium, 8 minutes for large

LOC 2 and higher: Also, correct/incorrect selection of camera as for Task 4.3a

Advancement Criteria: Student performs three trials consecutively without error and within time constraints per LOC.

#### **4.6 UNBERTHING**

Unberthing is a complex task. As with berthing, it requires a combination of skills which includes skill in using the hand controllers simultaneously, camera use skills, and visualization skills. It also requires a high degree of accuracy and a constant concern for safety.



#### Task 4.6a: Unberthing a Payload; Camera Skills

A grappled, berthed payload is depicted in the payload bay. (This is a small-sized payload with the grapple fixture on the port side.) The student must unberth the payload by lifting it straight up to low hover position. Visual aid: Lines are graphically displayed on the screen which are perpendicular to a line through the midpoint of the bottom of the V-guide and tangent to the bottom of the V-guide.

In this task, camera skills are also evaluated in LOC 2. The student must use the most appropriate camera(s) and adjust the camera(s) within an acceptable range for berthing the payload.

##### Advance Organizer

"The purpose of this task is to help you develop your ability to UNBERTH a payload. Like berthing, unberthing is a complex task that requires a combination of skills, which include skill in using the hand controllers simultaneously, camera usage, and visualization. It also requires a high degree of accuracy and a constant concern for safety.

"In this task, lift the payload straight up out of the V-guides to low hover position. At first, the cameras are set up in optimal positions, although you can change them if you wish. Very quickly, you'll see that you must actually select and adjust them yourself.

"As you perform this task, remember that the P2T2 allows structures to "pass through" one another visually; nevertheless, proper performance does not allow this! If the payload trunnions pass through the V-guides, you would probably have "damaged" either the payload, the Orbiter, or both.

"This task is evaluated for safety, accuracy, efficiency, and camera usage. SAFETY dictates that no part of the arm or payload come in contact with any other structure; in addition, you must use Vernier rate within ten feet of the payload. ACCURACY is a measure of tolerance for the clearance between the trunnions and V-guides. In normal RMS usage, you are allowed to "scrape" the V-guides with the trunnions; however, in this exercise, you are not. EFFICIENCY is a function of how directly you move the arm/payload to the target position, using both hand controllers simultaneously, and of time (when used).

"This task is divided into three levels of complexity. The first level of complexity gives you a ghost guide bar to follow and the cameras automatically reset for you. Be aware of the views given so you can recreate them. After the first level of complexity, camera skills are evaluated by your selection and adjustment. The higher levels of complexity will ask you to adjust one or both of the camera views, and to set up the RMS for unberthing. Time varies per level of complexity and begins when you move the hand controllers.

"When you are satisfied that you have successfully finished the task, use the mouse cursor to press the "Done" button."

**HINT buttons: Gods-eye View Buttons:** These buttons would give a gods-eye view of the shuttle for 60 seconds at a time each. Student starts with eight buttons (an editable number). (See task 4.5a.)

**Line Aid Buttons:** Like the Gods-eye View buttons, these buttons would allow the student to return the visual aid lines for 60 seconds each. Student starts with 5 buttons (an editable number), which don't appear until LOC 2.

Levels of Complexity:

- (1) As above
- (2) Remove the visual aid; start the cameras in "vanilla" position
- (3) Change the position of the grapple fixture (top, starboard). (Unless LOC 2 is implemented, remove visual aids except godseye at this LOC and start with cameras in "vanilla" position.)
- (4) Use larger payloads (medium and large)

**Evaluation:** As for Task 4.5a except for time limits. Reduce LOC 1 time limit by 1 minute; reduce all other time limits by 2 minutes

**Trunnions** do not have to be in the middle of the V-guides for high accuracy.

**Efficiency:** Student should move the arm within 115% of the shortest possible path. Hand controllers should be used simultaneously 50% of the time.

**Accuracy:** Accuracy is based on the student getting the arm to low hover position (like the earlier short fly-to tasks).

**Safety:** Every time the student hits the V-Guides with the trunnions, he/she gets -15 points from possible 100. Student must get 75 points or more to pass safety.

**Advancement Criteria:** Student performs three trials consecutively without error and within time constraints per LOC.

#### 4.7 FLY-TO POSITION-AND-ATTITUDE

This task, also known as "arm phasing," is a complex task. It is highly dependent upon visualization skills. Depending on the complexity of the task, it also requires camera skills and use of other information such as that found on Spec 96. It can be divided into unloaded-arm and loaded-arm phasing. Loaded-arm phasing is more complex because of the increased safety hazard of moving a payload near Orbiter structures.

##### Task 4.7a: Unloaded Arm Phasing; Camera Skills

The P2T2 is configured so that one window shows panel A8, one window is used for the performance indicator and feedback, and the other two windows show vanilla views of the arm. The task begins with the End Effector at a predetermined position and attitude. The student must place the End Effector at a different specified position and attitude of minimal complexity (no singularities or reach limits). The destination position and attitude is indicated by a wire-frame "ghost" arm in the target configuration: the student must line up the full-graphic arm with the ghost arm. Student uses a DONE button to indicate a completed job.

##### Advance Organizer:

"The purpose of this task is to give you more experience in flying the arm. In this task, the fly-to's are longer than in other P2T2 tasks you may have performed. These kind of longer fly-to's are also known as "arm phasing."

"Arm phasing (in this case, unloaded arm phasing) is a complex task due to its dependence on many different skills. Here, you will use planning and visualization skills to avoid reach limits and singularities while you move the arm to target position and attitude. Camera skills are important to monitor arm movement for safety. You may also use other information such as that found on Spec 96.

"In the early levels of complexity, a ghost arm is used to indicate the target position and arm attitude; later, the goal is to fly the arm to a set of target coordinates. The task is divided into five levels of complexity. The first level of complexity requires only THC inputs; the second level of complexity requires only RHC inputs; the remaining levels of complexity require simultaneous hand controller movements and will take away some of the earlier cues, so you may need to make visual and operational adjustments as well.

"This task is evaluated for accuracy, efficiency, safety, and camera usage. ACCURACY is a function of how close the arm is to the target position. EFFICIENCY is based using both hand controllers simultaneously, economy of movement (i.e., arm trajectory), and on time when used. SAFETY is a measure of avoidance of reach limits and singularities, and of making sure Vernier rate is used when within ten feet of the shuttle. CAMERA usage involves selecting and adjusting cameras to monitor the arm.

"When you are satisfied that you have successfully finished the task, use the mouse cursor to press the "Done" button."

**HINT buttons: Gods-eye View Buttons:** Student starts with eight buttons (editable to a different number). (See task 4.5a.)

**Ghost Arm Button:** Starting with LOC 3, student gets an initial set of 5 buttons (editable), which disappear when used. When a button is pressed, the target position is displayed using the ghost arm. This view lasts for 30 seconds.

**Levels of Complexity:**

- (1) As above. Start with movements requiring only THC inputs; new position/attitude is close to initial position
- (2) As above but using only RHC inputs; new position/attitude is close to initial position
- (3) As above using both hand controllers; new position/attitude is close to initial position
- (4) Target position is more difficult to achieve
- (5) Task is repeated without the wire-frame arm; student must use a set of target coordinates
- (6) Have a reach limit and/or singularity between initial position/attitude and target position; hitting a reach limit or singularity counts as an error and trial is reset using the same target coordinates. (Remove wire frame arm if LOC 5 is not implemented.)

**Evaluation:**

**Efficiency:** Student should move the arm within 115% of the shortest possible path. Hand controllers should be used simultaneously 75% of possible time. Time limits when used: For LOC 1, 2, and 3: 1 minute; LOC 4 and 5: 2 minutes; LOC 6: 5 minutes.

**Accuracy:** Allow 10% deviation per measure

**Camera skills:** as usual (25% accuracy)

**Advancement Criteria:** Student performs three trials in each LOC consecutively without error and within time constraints.

### Task 4.7b: Loaded Arm Phasing; Camera Skills

P2T2 is configured as for task 4.7a. The task begins with a small payload at a predetermined position and attitude. The student must place the payload POR at a specific position and attitude of minimal complexity. Specific payloads are picked randomly. (This is not an ungrappling task.)

#### Advance Organizer:

"The purpose of this task is to give you more experience in flying the arm, this time with a payload attached. Flying a payload with the arm is also known as "loaded arm phasing."

"Loaded arm phasing is more complex than unloaded because of the increased safety hazard of moving a payload near Orbiter structures. Once again, you will use planning and visualization skills to avoid reach limits and singularities while you move the arm to the target position and attitude. Camera skills are important to monitor arm movement for safety. You may also use other information such as that found on Spec 96.

"In the early levels of complexity, a ghost arm is used to indicate the target position and arm attitude. Later the goal is to fly the arm to a set of target coordinates. This task is divided into five levels of complexity. The first level of complexity requires only THC inputs; the second level of complexity requires only RHC inputs; the remaining levels of complexity require simultaneous hand controller movements and will take away some of the earlier cues, so visual and operational adjustments may need to be made.

"This task is evaluated for accuracy, efficiency, safety, and camera usage. ACCURACY is a function of how close the arm is to the target position. EFFICIENCY is based on using both hand controllers simultaneously, economy of movement (i.e., arm trajectory), and on time, when used. SAFETY is a measure of avoidance of reach limits and singularities, and of making sure the rate is vernier when within ten feet of the shuttle. Camera usage involves selecting and adjusting camera to monitor the arm.

"When you are satisfied that you have successfully finished the task, use the mouse cursor to press the "Done" button."

HINT buttons: Gods-eye View Buttons: Student starts with eight buttons (an editable number). (See task 4.5a.)

Ghost Arm Button: For LOC 3 and 4, student gets an initial set of 5 buttons (an editable number), which disappear when used. When a button is pressed, the target position is displayed using the ghost arm. This view lasts for 30 seconds.

Levels of Complexity:

- (1) As above
- (2) Target position is more difficult to achieve
- (3) Task is repeated without the wire-frame arm; student must use a set of target coordinates
- (4) Larger payloads are used (one medium; then one large)
- (5) Have a reach limit and/or singularity between initial position/attitude and target position

Evaluation:

Efficiency: 50%: Student should move the arm within 115% of the shortest possible path. 40%: Simultaneous use of hand controllers: 50% of possible time. 10%: Time limits when used, for LOC 1, 2 and 3: 1 minute; LOC 4: 3 minutes; LOC 5: 3 minutes for medium, 4 minutes for large

Accuracy: Allow 10% deviation per measure

Camera skills: as usual (25% accuracy)

Advancement Criteria: Student performs three trials in each LOC consecutively without error and within time constraints.

#### 4.8 DEPLOYMENT

Deployment is a complex task that involves several subtasks: fly-to positions, grappling, unberthing, and ungrappling.

##### Task 4.8a: Whole Task: Deployment

The P2T2 is configured with panel A8, a window for the performance indicator and feedback, and two windows with vanilla views of the arm. (Either of the two arm views can be replaced by the student with the typical aft window view of the shuttle.) The IBSS (SPASII) payload is berthed in the bay and the arm is in the pre-cradle position. The student must deploy the payload to its flight-specific position-and-attitude and return the arm to the pre-cradle position using the appropriate mode during each phase of the task. The student is evaluated on safety, efficiency, accuracy, and procedural correctness. In addition, camera usage is evaluated.

**Advance Organizer:**

"The purpose of this task is to help integrate everything you have learned so far about performing RMS tasks. Your mission is to deploy the SPAS payload. This task uses actual flight-specific positions as found in the PDRS Operations Checklist P2T2 Flight Supplement "Generic Procedures."

"Deployment is a complex task that involves several subtasks: fly-to positions, grappling, unberthing, and ungrappling. In addition, such factors as mode of operation, use of coordinate systems, rate of movement, camera usage, and all aspects of safety (including avoiding reach limits) are involved.

"Your task is to fly the arm from pre-cradle position to pre-grapple; to grapple and unberth the payload; to deploy the payload at the target position; and to return the arm to pre-grapple. At first, the deploy position will be indicated by a ghost arm; but then you will use target coordinates to determine this yourself.

"Be sure to create a mental plan before you begin. Imagine in detail how the arm will need to reconfigure as it moves to the deploy position. Consider carefully what your problems are likely to be. How can you avoid these problems? How can the cameras help you? What other information can you use? Remember to monitor the arm as you perform the deployment task, and revise your plan as needed.

"In this task, you are evaluated on safety, efficiency, accuracy, camera usage and procedural correctness. When you are satisfied that you have successfully finished the task, use the mouse cursor to press the "Done" button."

**Levels of Complexity:**

(1) As above.

**Evaluation:**

Most subtask components should be evaluated as they were for the part tasks. Some composite score can then be created for each measure (e.g., safety, or efficiency, or accuracy, etc.).

When the timer is used: Entire task must be completed within 15 minutes.

Singularities and reach limits: If the student hits a singularity or reach limit, do not count this as an error; however, 5% of the score for efficiency and 10% of the score for safety should be taken off. Notify the student that points have been deducted. Student must then be able to recover or an error has been made.

Software stops: If the student flies the arm into a software stop, both efficiency and safety scores should be halved. Notify the student that points have been deducted. Student must then be able to recover or an error has been made.

Camera skills: evaluated as usual

Advancement Criteria: Student must perform three consecutive trials within usual time limits where applicable.

#### 4.9 RETRIEVAL

Retrieval is a complex task that involves several subtasks: fly-to positions, grappling, berthing, and ungrappling.

##### Task 4.9a: Whole Task: Retrieval

The P2T2 is configured with panel A8, a window for the performance indicator and feedback, and two windows with vanilla views of the arm. (Either of the two arm views can be replaced by the student with the typical aft window view of the shuttle.) The SPAS-I payload is in orbit and the arm is in the pre-cradle position. The student must retrieve the payload, berth it, and return the arm to the pre-cradle position using the appropriate mode during each phase of the task. The student is evaluated on safety, efficiency, accuracy, procedural correctness where applicable, and camera skills.

##### Advance Organizer:

"The purpose of this task is to help integrate everything you have learned so far about performing RMS tasks. Your mission is to retrieve the SPAS-I payload using actual flight-specific data and procedures (see PDRS Operations Checklist P2T2 Flight Supplement "Generic Procedures").

"Retrieval is a complex task that involves several subtasks: fly-to positions, grappling, unberthing, and ungrappling. In addition, such factors as mode of operation, use of coordinate systems, rate of movement, camera usage, and all aspects of safety (including avoiding reach limits) are involved.

"Your task is to fly the arm from pre-cradle position to pre-grapple; to retrieve and berth the payload; and to return the arm to pre-grapple.

"Be sure to create a mental plan before you begin. Imagine in detail how the arm will need to reconfigure as it moves to the retrieval position and then moves to berth the payload. Consider carefully what your problems are likely to be. How can you avoid these problems? How can the cameras help you? What other information can you use? Remember to monitor the arm as you perform the retrieval task, and to revise your plan as needed.



"In this task, you are evaluated on safety, efficiency, accuracy, and procedural correctness. In addition, camera usage is evaluated.

"Use the DONE button if you finish before the timer runs out (if the timer is used)."

Levels of Complexity:

(1) As above.

Evaluation:

Most subtask components should be evaluated as they were for the part tasks. Evaluation should keep the individual scores (e.g., safety, or efficiency, or accuracy, etc.); student must master each measure.

When the timer is used: Entire task must be completed within 15 minutes.

Singularities and reach limits: If the student hits a singularity or reach limit, do not count this as an error; however, 5% of the score for efficiency and 10% of the score for safety should be taken off. Notify the student that points have been deducted. Student must then be able to recover or an error has been made.

Software stops: If the student flies into a software stop, scores for both efficiency and safety should be halved. Notify the student that points have been deducted. Student must then be able to recover or an error has been made.

Camera skills: evaluated as usual

Advancement Criteria: Student must perform three consecutive trials without error and within usual time limits where applicable.

#### **4.10 UNCRADLE/CRADLE**

##### **Task 4.10a: Uncradle**

Uncradling the arm is generally the first task an arm operator must perform. This task is a very specific procedural task.

The student sees the normal P2T2 windows including panel A8 and the tutoring window. The student must simply perform the uncradle procedure to move each joint to the required position. The uncradle procedure is:

1. Place arm in single joint mode; use Vernier rate
2. Wrist pitch: move the wrist out of the aft MPM until it is pitched up at 5 degrees

3. Elbow pitch: move the elbow out of the mid MPM until it is pitched up at 1 degree
4. Shoulder pitch: Move the shoulder out of the forward MPM until it is pitched up at 25 degrees
5. Bring the elbow pitch down to -25 degrees

Advance Organizer:

"The purpose of this task is to practice the procedure for uncradling the arm.

"Uncradling the arm requires use of SINGLE mode and vernier rate. Use the readouts on panel A8 as you move the various joints out of the MPMs to their proper "pre-cradle" position.

"This task is evaluated for accuracy (procedural correctness), efficiency, and safety.

"When you have completed the uncradle procedure and the arm is in the pre-cradle position, press the NEXT TASK button."

Levels of Complexity: NA

Evaluation: This task is evaluated for accuracy (procedural correctness), safety, and efficiency. Safety is a function of using vernier rate. The student has performed efficiently if he/she did not overshoot the target values. When used, time allowed is 60 seconds and counts for 1/3 of the efficiency score.

Advancement Criteria: Student must perform three consecutive trials without error and within time limit.

#### Task 4.10b: Cradle

Cradling the arm is generally the final task an arm operator must perform. Like uncradling, this task is a very specific procedural task.

The student sees the normal P2T2 windows including panel A8 and the tutoring window. The arm should start in various positions and attitudes. The student must simply perform the cradle procedure by first moving the arm to the precradle position (which is the same as the resulting position of the uncradle procedure), then lowering each joint into the respective MPMs. The cradle procedure is:

1. Place arm in single joint mode; use Vernier rate

2. Bring the elbow pitch to -25 degrees, shoulder to 25 degrees, wrist to 5 degrees. Arm should be in line with the longeron. (This is called pre-cradle position.)
3. Elbow pitch: Move the elbow until it is pitched up at 1 degree
4. Shoulder pitch: Move the shoulder until it is at zero degrees; this puts it into the forward MPM
5. Elbow pitch: Move the elbow from 1 degree to zero; this brings it into the mid MPM
6. Wrist pitch: Move the wrist from 5 degrees to zero; this places it in the aft MPM and completes the procedure.

Advance Organizer:

"The purpose of this task is to practice the procedure for cradling the arm.

"Cradling the arm requires use of SINGLE mode and vernier rate. In this task, you must first move the arm to the pre-cradle position before you actually cradle the arm. Use the readouts on panel A8 to make sure the arm is in the proper pre-cradle position before you actually cradle the arm.

"This task is evaluated for accuracy (procedural correctness), efficiency, and safety.

"When you have completed the cradle procedure, press the NEXT TASK button."

Levels of Complexity: NA

Evaluation: This task is evaluated for accuracy (procedural correctness), safety, and efficiency. Safety is a function of using vernier rate. The student has performed efficiently if he/she did not overshoot the target values. When used, time allowed is 60 seconds and counts for 1/3 of the efficiency score.

Advancement Criteria: Student must perform three consecutive trials without error and within time limit.



## APPENDIX A: CHANGING THE EDITABLE PARAMETERS

### OVERVIEW

In this section you will find step-by-step instructions on how to change the editable parameters as well as a discussion of what the editable parameters are. Changing the parameters is not difficult, but it requires careful attention to what you're doing, since the editable parameter files are used by the Intelligent Trainer when running the tasks. If you corrupt the editable parameters files, the trainer cannot function correctly, and in fact, may crash entirely.

### TYPES OF EDITABLE PARAMETERS AND WHERE TO FIND THEM

There are two types of editable parameter files: global and task-specific.

Global parameters: A global parameter affects all tasks. An example is use of the timer, which can be "turned off" at the global level and is not then used in any task. This saves you from having to turn the timer off for every part task.

The global parameters file is found under:

.../data/part\_task\_data/ /globalParams/editableParameters.dat

(Note that the Silicon Graphics computer distinguishes between capital and lowercase letters.)

Task-specific parameters: A task-specific parameter affects only that particular part task. An example of such a parameter is the amount of time allowed for a student to do the particular task. Every part task and each of the two whole tasks have their own editable parameters file.

The task-specific editable parameters files can be found under:

.../data/part\_task\_data/ <name of part task> /editableParameters.dat

where <name of part task> might be "grapple", "unberth", etc. (Note that the Silicon Graphics computer distinguishes between capital and lowercase letters.)

### WHAT PARAMETERS ARE EDITABLE?

Appendix B provides a listing of the editable parameters for both global and task-specific parameters.

## CHANGING EDITABLE PARAMETERS

The following steps tell you how to change an editable parameter. The steps assume no knowledge of the Silicon Graphics computer.

The Silicon Graphics machine can show you files and directories in two modes: as text and as labeled graphic icons. Since it's easiest to use the icon mode, the first step below tells you how to change to this mode if it's not already being used. Also, you may need to find out from the system administrator how to get to the top-level P2T2\_IT directory (e.g., /usr/global), since that part would vary from machine to machine.

When in icon-display mode, you use the mouse to tell the computer what to do. "Clicking" on an icon means putting the mouse cursor (a little arrow) on the icon and clicking the left button, usually twice in rapid succession. Sometimes a different mouse button is used for other tasks, but this is spelled out for you in the steps when applicable.

PROCEDURE: CHANGING EDITABLE PARAMETERS

---

STEP	ACTION
1	<p>From the Console window, if the files and directories are NOT already displayed as icons, change to icon mode, which is an easier way to perform tasks:</p> <p>Type in <b>dirview</b> and the entire pathname of the P2T2 IT directory through the <b>part_task-data</b> directory and press the <i>Enter</i> key. This will be something like:</p> <p><b>dirview .../P2T2_IT/data/part_task_data</b></p> <p>The first part of the path (represented with three dots) might be <b>/usr/global/</b> and depends on how things are organized in your computer. See the system administrator if you have any problems.</p>
2	<p>Select the <i>data</i> directory:</p> <p>Click twice on the icon labeled <i>data</i></p>
3	<p>Select the <i>part_task_data</i> directory:</p> <p>Click twice on the <i>part_task_data</i> icon</p>
4	<p>Select the directory for the part task you wish to edit:</p> <p>Click twice on the icon for the appropriate part task</p>

---

PROCEDURE: CHANGING EDITABLE PARAMETERS

---

STEP	ACTION
5	<p>Make a copy of the editable parameters file in case the original gets corrupted:</p> <ul style="list-style-type: none"><li>a) Put the cursor on the <i>editableParameters.dat</i> file and click the LEFT button <u>once</u>.</li><li>b) Hold down the RIGHT mouse button. A little menu will appear.</li><li>c) Continuing to hold down the right mouse button, move the mouse cursor down the menu to the <i>Make New Copy</i> option. With the cursor on the <i>Make New Copy</i> option, release the mouse button.</li></ul> <p>You will notice a new file is created called <i>copy.of.editableParameters.dat</i>. It is recommended that you do not delete this file.</p> <p>If you get a message that says "Cannot create files in this directory," have the system administrator change the file permissions. <i>You will not be able to change your parameter edits either until this has been done.</i></p>
6	<p>Select the <i>editableParameters.dat</i> file in order to change the parameter(s):</p> <p>Click twice on the <i>editableParameters.dat</i> icon</p>
7	<p>Play with the scrollbars on the left and bottom of the file window to see how to "get around" in the file.</p> <ul style="list-style-type: none"><li>. Try clicking once at different points in the "scrollbar window"</li><li>. Click once on the little up and down arrows at the bottom (and the left) of the scrollbar windows</li><li>. "Drag" the scrollbar up or down (put the mouse cursor on the scrollbar itself, hold down the left button and, continuing to hold down the left button, move the mouse cursor up or down).</li></ul>



---

PROCEDURE: CHANGING EDITABLE PARAMETERS, cont'd

---

STEP	ACTION
8	Scroll through the file until you get to the parameter you wish to change.
9	Change the parameter value:  a) Put the mouse cursor to the right of the parameter <i>value</i> and click once; a blue vertical bar appears, which is the text-editing cursor.  DO NOT CHANGE THE NAME OF THE PARAMETER, for example, AccuracyWgt. Change only the <i>value</i> of the parameter, for example, 50.  b) Use the Back Space key to erase the old parameter value.  c) Type in the new value.
10	Save your data and exit the editableParameters.dat file:  With mouse cursor in the editableParameters.dat window, hold down the RIGHT mouse button. While still holding down the right button, move the cursor down to the Quit option. When you release the mouse button, a dialogue window appears; select Save & Quit. Your changes will be saved and you will return to the part_task_data directory.  (You can also simply select Quit from the dialogue window, which, unless you have just Save-d your changes, will cause your changes to be lost.)
11	Repeat steps 4-10 above if you wish to edit parameters for other tasks.
12	To close windows:  Click once on the small square (a lightening-bolt icon) in the upper right corner of each window you wish to close.

---

---



## APPENDIX B: A Listing of the EDITABLE PARAMETERS

### OVERVIEW

In this section, you will find a listing of the editable parameters files. Reviewing this section will help you see how the P2T2 Intelligent Trainer can be tuned or modified to suit your training data or requirements.

### NOMENCLATURE

In the file printouts below, the pound sign (#) is a signal to the computer to ignore what follows. This is used in order to document things for human beings.

The actual names and values of editable parameters are NOT preceded by a pound sign. These are the lines the computer actually reads in order to get the values for the particular parameters and make the P2T2 Intelligent Trainer work.

Consider an example:

---

```
## Variables for the number of trials in each level of complexity (LOC).  
##  
LOC1tasks 5
```

---

In the example above, the line "## Variables for the number of trials..." (etc.) is a comment to you, the reader, and documents what follows. The word "LOC1tasks" is the name of the editable parameter, and must not be changed! The number "5" is the value of the parameter LOC1tasks, and is what you change in order to change the number of trials the student gets in the first level of complexity for this part task.

Please read Appendix A before attempting to change any editable parameters in the P2T2 Intelligent Trainer software.

### EDITABLE PARAMETERS (PRINTOUTS)

#### Global parameters

```
## Variables for the number of trials in each level of complexity (LOC).  
##  
LOC1tasks 5  
LOC2tasks 5  
LOC3tasks 5  
LOC4tasks 5
```

```
##  
## Variables for the time (in seconds) within which the student must  
## finish the task in order to receive a passing time score.  
##
```

LOC1time 30  
LOC2time 60  
LOC3time 90  
LOC4time 90

##

## For Payload Tasks:

##

LOC1time 60  
LOC2time 90  
LOC3time 120  
LOC4time 120

##

## For OrbLd Tasks:

##

LOC1time 120  
LOC2time 180  
LOC3time 240  
LOC4time 240

##

## Variables for the various weights given to the different evaluations.

## The weights are expressed as percentage of the total evaluation.

##

##

## Final evaluation weights. Accuracy weight is as listed and efficiency  
## weight is 1-AccuracyWeight ("one minus the value of AccuracyWgt").

##

AccuracyWgt        0.75

##

## Efficiency evaluation weights. Path weight is as listed, and time weight  
## is 1 - path weight ("one minus the path weight").

##

PathWgt            0.80

## fly-to, berth, unberth, Ld + UnL Arm Phasing

##

## NumGodsEyes is the number of god's eye views to give the student in  
## higher levels of complexity. NumGhostArms is the number of ghost arms  
## the student can access in the last level of complexity. The ghost arm  
## depicts the goal arm, providing a visual aid for the student.

##

NumGodsEyes        6

NumGhostArms       4

Appendix B: A Listing of the Editable Parameters

---

## Efficiency evaluation weights. Path weight is as listed, and time weight  
## is 1 - path weight ("one minus the path weight").

##  
PathWgt 0.80

## Short fly-to tasks, Berth, Unberth, LD and UNL Arm Phasing

##  
## NumGodsEyes is the number of god's eye views to give the student in  
## higher levels of complexity. NumGhostArms is the number of ghost arm views  
## the student can access in the last level of complexity. The ghost arm  
## depicts the goal arm, providing a visual aid for the student.

##  
NumGodsEyes 6  
NumGhostArms 4

##  
## Berth, Unberth part tasks

##  
numLineAids 5

## Recognizing Singularities and Reach Limits, Visualization of Singularities and Reach  
Limits (four part tasks)

##  
## This variable determines the number of times the student can view the  
## hint window, which contains the definition of reach limits in each joint.  
## The definition is at the end of this data file.

##  
numHints 4

## Short fly-to tasks, Ungrapple, Grapple, Berth, Unberth, LD + UNL Arm Phasing, and  
## Visualization of Singularities and Reach Limits

##  
## Path Evaluation Editables

##  
## The path evaluation limits are used to compute scores for path  
## evaluations in this part task (which are parts of the efficiency score).  
## They are used as follows: for any traversal between two points, there  
## is a minimum path. If the student uses the minimum path, the score  
## is 100. If the path length is greater than the minimum path times  
## the pathDistanceLimit, then the path score is below the passing score.

##  
pathDistanceLimit 1.76  
pathRotationLimit 1.76

## Grapple, Berth

##

## Accuracy Evaluation Editables

##

## TranslationAllowance is the maximum translational error allowed in the task. Greater errors will result in a failing accuracy score.

## For grappling, the goal position is the base of the grapple plate, and acceptable performance is when the arm has reached the grapple envelope, which is 8 inches from the base of the grapple plate.

##

TranslationAllowance 8.0 [Grapple, Berth]

RotationAllowance 2.0 [Berth]

## Ungrangle, Grapple, Berth, Unberth, LD + UNL Arm Phasing

##

## Hand Controller Usage Evaluation Editables

##

## Following parameters are used to evaluate the student's efficiency in using the hand controllers. These parameters are only applicable in integrated mode.

## Given different amounts of rotation and translation, depending on how much time each requires, there is a best possible percentage of the total time during which the student can use both hand-controllers simultaneously. For example, if the rotational movement takes 20 seconds to complete and the translational movement takes 40 second to complete, the best possible hand-controller usage is 50%. HC\_usage\_ratio defines a fraction of that number. If the student does not use both hand-controllers more than that fraction of the best usage, the path score will be reduced. As in the previous example, the best HC usage is 50% of the time, and if HC\_usage\_ratio is defined to be 60%, then the student must use both hand-controllers at least  $50\% \times 60\% = 30\%$  of the time in order to avoid the penalty.

## HC\_penalty defines the percentage of the path score deducted if the above criteria are not met.

##

HC\_usage\_ratio .5

HC\_penalty .4

## Ungrangle, Grapple, Berth, Unberth

##

## Procedure Evaluation Editables

##

## When the student commits a procedure error, an amount of points equal to procErrDeduction is subtracted from the procedures score, which starts at 100.

##

procErrPenalty 20

## Ungrapple, Grapple

##

## Camera Evaluation Editables

##

## EECamRatio is the amount of time the EE camera should be used when moving  
## the arm from pregrapple to the grapple envelope. A value of .9 means that  
## the student should use the EE camera 90% of the time when the arm is  
## moving.

##

## EEGFcamRatio is the amount of time the grapple fixture and the End Effector  
## should be in view by the same camera while the arm is still moving.

## Not only should they be in view, but they must be zoomed in close  
## enough to see what is going on.

##

## camZoomRatio is how far a camera must be zoomed in to view the end  
## effector. The number is the percentage of the window filled by the end  
## effector. Increasing camZoomRatio will force the student to zoom in  
## closer, decreasing it will allow the student to zoom out more.

##

EECamRatio .9

EEGFcamRatio .9

camZoomRatio .007

## Berth, Unberth

##

## Camera Evaluation Editables

##

## vGuideZoomFactor is how far a camera must be zoomed in to view a V-Guide  
## while the payload is being lifted. The number is the percentage of the  
## window that must be filled by the V-Guide. Increasing vGuideZoomFactor  
## will force the student to zoom in closer.

##

## camViewRatio is the percentage of time the cameras must be in a good view.  
## At first the camera must be zoomed in to view the V-Guide to monitor  
## collisions. Once the trunnions clear the V-Guides, the camera must be  
## viewing the arm or the payload.

##

vGuideZoomFactor 0.027

camViewRatio 0.90

## LD and UNL Arm Phasing

##

## Camera Evaluation Editables

##

## minCamViewRatio is the ratio of time that both cameras should be viewing  
## a significant area. Significant areas include any of the arm joints,  
## the point of resolution, and the goal point of resolution. If a  
## camera is not viewing a significant area minCamViewRatio of the  
## time, then it is an error. For example, if minCamViewRatio is .85, then  
## the student must be viewing significant areas 85% of the time.

##

## cam90degRatio is a measure of how often the student should be viewing from  
## cameras that are on opposite sides of the object being viewed. This  
## is so the student can get better 3D views of the object. Generally  
## the student should try to view through 2 cameras that are 90 degrees  
## different from the object being viewed, although many times the  
## student will not be trying to view the same object.

##

## "Time" here is not a measure of time elapsed, but a measure of how many  
## times the arm is in view when the arm position changes. In effect it is  
## measure of the time the arm is in view while it is moving.

##

minCamViewRatio .85

cam90degRatio .50

## Ungrapple, Grapple, Berth, Unberth, Deploy, Retrieve

##

## Safety Evaluation Editables

##

## The following variables define the amount of points deducted from  
## the overall safety score (starts at 100) for each safety violation.

##

## collisionPenalty is subtracted from the safety score each time the arm  
## collides with the grapple plate or the grapple pin.

## coarseRatePenalty is subtracted each time the arm is moved in coarse  
## rate while within vernRateInFeet of the shuttle. vernRateInFeet is  
## a global editable parameter that specifies how far away the POR  
## should be from the shuttle before coarse rate is allowed.

## directModePenalty is subtracted from the safety score each time DIRECT  
## mode is entered.

## armConfigPenalty is subtracted each time the arm goes into a reach limit  
## or a singularity.

##

collisionPenalty 25.0

coarseRatePenalty 4.0

directModePenalty 10.0

armConfigPenalty 10.0



```
## LD and UNL Arm Phasing
##
## Safety Evaluation Editables
##
## The following variables define the amount of points deducted from
## the overall safety score (starts at 100) for each safety violation.
##
## coarsePenalty is subtracted each time the arm is moved in coarse
## rate while within vernRateInFeet of the shuttle. vernRateInFeet is
## a global editable parameter that specifies how far away the POR
## should be from the shuttle before coarse rate is allowed.
## directModePenalty is subtracted from the safety score each time DIRECT
## mode is entered.
## armConfigPenalty is subtracted each time the arm goes into a reach limit
## or a singularity.
##
armConfigPenalty 100
coarsePenalty      5
directModePenalty 10

## All tasks
##
## Advance organizer text that appears before starting the part task.
##
advOrganizer ""

## Visualization of Singularities
##
## Deductions - points subtracted from the accuracy score, which start at 100
##
## Identification Deduction is subtracted from the accuracy score for each
## extra attempt (after the first) at identifying the type of singularity.
##
## HC action Deduction is subtracted from the accuracy score for each
## extra attempt at identifying the hand-controller action needed to drive
## the arm shown to the singularity.
##
## First HC Action Deduction is the same as the above except it's used as
## deduction if the first attempt failed. Due to the ambiguous nature of
## this section of the task (more than one hand-controller action can drive
## the arm into the reach limit), First HC Action Deduction should be
## less than the standard HC_Action_Deduction.
##
## Singularity Not Reach Deduction is subtracted from the accuracy score
## if the student failed to drive the arm into the singularity.
##
## Singularity_Not_Resolved_Deduction is subtracted from the accuracy score
```

```
## if the student failed to resolve the singularity or if the final arm
## is too far off from the goal arm.
##
## Goal Not Reached Deduction is subtracted from the accuracy score
## if the student was unable to drive the arm to the goal configuration
## after the singularity had been resolved.
##
Identification Deduction          25.0
HC Action Deduction              15.0
First HC Action Deduction        5.0
Singularity Not Reached Deduction 20.0
Singularity Not Resolved Deduction 20.0
Goal Not Reached Deduction       20.0
##
## Penalties - percentage of the total accuracy score subtracted
##
## Hit Reach Limit Penalty is the percentage of the accuracy score
## subtracted if the student hit a reach limit while driving the arm.
##
## Hit Software Stop Penalty is the percentage of the accuracy score
## subtracted if the student hit a software stop while driving the arm.
##
## Hit Hardware Stop Penalty is the percentage of the accuracy score
## subtracted if the student hit a hardware stop while driving the arm.
##
Hit Reach Limit Penalty          0.50
Hit Software Stop Penalty        0.75
Hit Hardware Stop Penalty        1.00
##
##
## Visualization of Reach Limits
##
## Deductions - points subtracted from the accuracy score, which start at 100
##
## Identification Deduction is subtracted from the accuracy score for each
## extra attempt at identifying the location of the reach limit.
##
## HC action Deduction is subtracted from the accuracy score for each
## extra attempt at identifying the hand-controller action needed to drive
## the arm shown to the reach limit.
##
## First HC Action Deduction is the same as the above except it's used as
## deduction if the first attempt failed. Due to the ambiguous nature of
## this section of the task (more than one hand-controller action can drive
## the arm into the reach limit), First HC Action Deduction should be
## less than the standard HC Action Deduction.
##
```

Appendix B: A Listing of the Editable Parameters

## Reach Limit Not Reach Deduction is subtracted from the accuracy score  
 ## if the student failed to drive the arm into the reach limit.

##  
 Identification Deduction 20.0  
 HC Action Deduction 15.0  
 First HC Action Deduction 5.0  
 Reach\_Limit\_Not\_Reached\_Deduction 35.0

##  
 ##  
 ##  
 ## Penalties - percentage of the total accuracy score subtracted

##  
 ## Hit Software Stop Penalty is the percentage of the accuracy score  
 ## subtracted if the student hit a software stop while driving the arm.

##  
 ## Hit Hardware Stop Penalty is the percentage of the accuracy score  
 ## subtracted if the student hit a hardware stop while driving the arm.

##  
 Hit Software Stop Penalty 0.50  
 Hit\_Hardware\_Stop\_Penalty 0.75

#####  
 #####

Global Parameters

#####  
 #####

##  
 ## Note: Global parameters affect all part tasks.

##  
 ##  
 ## Show-timer tells the system whether the timer in the performance indicator  
 ## should be shown. A value of 1 means the timer will be shown, and  
 ## a value of 0 means the timer will be hidden.

##  
 show-timer 1

##  
 ## Proficiency-trials is the number of trials the student must pass  
 ## consecutively on the first try in order to proficiency a level of  
 ## complexity. For example, a level of complexity contains 5 trials.  
 ## If the student fails a trial at any given point, he or she must start over  
 ## and attempt to pass 5 consecutive trials again. However, if the student  
 ## is able to pass the very first 2 trials (or, actually, the value of  
 ## proficiency-trials) consecutively, he or she can  
 ## proceed to the next level of complexity regardless the specific

## requirements of the previous level of complexity, 2 being the value  
## of "proficiency-trials." In order to turn off the proficiency-trial  
## mechanism, set the following value to a number greater than 100 or to 0.  
##  
proficiency-trials 2

##  
## passingScore defines the minimum score a student must get for an  
## evaluative score in order to pass. It is always between 0 and 100.  
## The success of EACH evaluation, including the final evaluation combining  
## accuracy, efficiency, safety, procedure and camera evaluation, is  
## determined by whether its score is greater than the passing score.  
##  
passingScore 75

##  
## The amount of time allowed for each task is defined in the part-task  
## editable-parameter files. The following variable provides a fraction  
## of that allowed time within which if the student completes the task, a  
## time score of 100 will be given. For example, a timeScoreFactor of 0.5  
## means that if the student completes a task within half of the allowed  
## time, he or she will receive a time score of 100.  
##  
timeScoreFactor 0.50

##  
## vernRateInFeet specifies how many feet the POR must be from the  
## orbiter before the student is allowed to switch the rate to Vernier.  
## This is used in many of the safety evaluations. For example, if  
## vernRateInFeet is 5.0, and the student moves the arm in coarse  
## rate while within 5 feet of the shuttle, then points are deducted from  
## the safety score for that trial. The number of points deducted per  
## violation varies per task, accordingly to the editable parameter in  
## that task that deducts points (it is called coarsePenalty or  
## coarseRatePen, or something similar).  
##  
vernRateInFeet 10.0

##  
## TranslationAllowance & RotationAllowance are used in calculating accuracy  
## scores. These will be overridden if the part tasks defines their own values  
## for these variables.  
##  
## TranslationAllowance determines within what distance, in inches, the  
## final position should be from the goal position for a successful trial.

Appendix B: A Listing of the Editable Parameters

---

## RotationPitchYawAllowance gives the acceptable error in degrees for pitch  
## and yaw, which is the maximum allowable angle from the goal-axis. This  
## evaluation can be pictured as defining a cone centered at the goal axis.  
## The final arm's axis must fall within the cone for a passing accuracy score.  
## RotationRollAllowance is the maximum allowable error (in degrees) for roll.  
##  
TranslationAllowance 5.0  
RotationPitchYawAllowance 8.0  
RotationRollAllowance 5.0

##  
## Following variables define the amount of time in seconds during which  
## a hint is showed on P2T2. These affect all part tasks in which these  
## forms of hint are implemented.  
##  
## GhostArm - a wire-frame arm that shows the target arm configuration  
## GodsEye - a god's eye view from space on significant elements  
## The above two types of hints are used in the short fly-tos,  
## arm phasing, berth and unberth tasks.  
##  
## Berthlines - a visual guide that erects vertically from the V-Guides.  
## Berthlines are used in the Berth and Unberth tasks.  
##  
GhostArmTime 45  
GodsEyeTime 45  
BerthLinesTime 45

##  
## demo-mode is a flag determining the mode of the system. When the system  
## is in demo mode, there is an extra button in each part task that allows  
## the student to go to the next task without having to go through the  
## current task. A value of 0 flags regular mode, and 1 flags demo mode.  
##  
demo-mode 1

##  
## dummy-its is a flag determining whether to skip through the part task  
## tests and remediation in maintenance mode. If dummy-its is 1, instead of  
## actually testing or remediating, a window comes up and it allows the user  
## to mark which parts of the test as mastered or misused. This is used  
## to demonstrate and to test the ITS portion of the program.  
##  
dummy-its 0



GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.

NASA SBIR Phase II Final Report

Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"

August 15, 1991

Appendix C: Programmer's Reference

---

<b>APPENDIX C: PROGRAMMERS REFERENCE</b>
--





---

<b>TABLE OF CONTENTS</b>
--------------------------

---

---

<b>SUBJECT</b>	<b>PAGE</b>
Overview of the P2T2 Intelligent Trainer Environment.....	2
Main Directory .....	2
Sub-Directories .....	2
ITS Executable Files .....	3
ITS Source Code .....	4
Overview .....	4
Descriptions of the src Directories .....	4
Makefiles.....	5
How to Change the ITS .....	6
Overview .....	6
Modifications to P2T2 .....	6
How P2T2 and ITS Communicate .....	6
How to Add New Commands/Data to the P2T2 ITS Interface .....	7
How to Integrate ITS Into a New Version of P2T2 .....	8
A Part Task Consists of Four Classes .....	9
Integrating the Part Task into the System.....	10
ITS Data Files .....	13
Part Task Data Directory and Editable Parameters .....	13
How to Add Editable Parameters .....	13
Payload Database.....	14
Arm Configuration Database.....	16
Domain Hierarchy Data File .....	18
Student Data Files.....	18
ITS <i>misc</i> Directory .....	19
Appendix A: ITS Source File List.....	A-1
Appendix B: The Formats of the Data Files.....	B-1
Appendix C: Modifications to P2T2 Source Code .....	C-1
Appendix D: Domain Hierarchy .....	D-1

---

<b>OVERVIEW OF THE P2T2 INTELLIGENT TRAINER (ITS) ENVIRONMENT</b>
---

### **Main Directory**

The P2T2 Intelligent Trainer (hereafter called ITS in this document, an acronym for "Intelligent Tutoring System") is located in the directory P2T2.IT, which contains everything necessary to run the ITS including the P2T2 simulation software itself. Since all path references in the system are relative to this root directory, the system can be copied anywhere else simply by copying the entire P2T2.IT directory.

### **Sub-Directories**

P2T2.IT consists of the following sub-directories:

- bin contains the executable binary files, including:
  - . modptt : a modified version of P2T2
  - . ptutor : the Intelligent Trainer
  - . pits : a control program that launches P2T2 and the ITS
  - . shell script files that start up the Intelligent Trainer
  - . CLIPS files containing procedural network rules and definitions
  - . default.ftt and pttdefaults : two P2T2 default files that define P2T2's default configuration
- data contains the data files ITS accesses during execution. These include files containing student information, editable parameters for each part task, a payload database, and an arm configuration database.
- misc contains supporting files needed by both P2T2 and ITS. Model files and various other graphical objects and image files are stored in this directory.
- src contains the source code for the executable programs in bin.

The contents of each directory is discussed in detail in this document.

## ITS EXECUTABLE FILES

### ITS Executable Files

To run ITS, several shell files in *bin* can be executed without having to worry about arguments passed to the binaries. These files are:

*keybd\_its* : runs the ITS and assumes you have only the keyboard, dials, and buttons

*chair\_its* : runs the ITS and assumes you have hand controllers as well as buttons and dials. Depending on which hand controllers you have, the simulator (P2T2) may have to be recompiled for the available HC chair.

*pits* is the program that executes P2T2 and ITS and sets up a message queue in which P2T2 and ITS exchange information. All the shell files make calls to *pits*. *pits*' command line has the following format:

```
pits -its [command to execute ITS] -modptt [command to execute P2T2]
```

The ITS executable is *ptutor*. (*ptutor* must be started up using *pits*.)

```
ptutor [filename]
```

where *filename* is the path and filename of the GIF image used in the introductory screen. The default is *../misc/gifs/discover.gif*.

The modified P2T2 is called *modptt*. (*modptt* must be started up using *pits*.)

```
modptt -c -w 3 -s
```

-c flags that a chair with hand controllers is installed

-w 3 indicates 3 windows are defined

-s flags the set-up mode, which enables the user to change the P2T2's configuration by accessing a menu using the right mouse button.

Note that neither *modptt* nor *ptutor* will run on their own: their message queues require that they both be active, so they must always be run concurrently.

## ITS SOURCE CODE

### Overview

The source code for the P2T2 Intelligent Trainer (ITS) is contained in a hierarchy of directories where each directory contains a logical sub-unit. Since the ITS is comprised of three separate programs (*pits*, *ptutor*, and *modptt*), the source code is distributed among three main directories along with other supporting directories.

src contains source code for the following:

- common : definitions used by both *modptt* and *ptutor* for communication
- pits : the startup driver program source code
- modptt : modified P2T2 source code
- ptutor : ITS source code
  - clips : CLIPS expert system shell
  - its : bulk of ITS source code
  - graphics : graphical objects
- image : code used to process images; used by both *modptt* and *ptutor*

Appendix A provides a list of all the source files of the *its* directory and brief descriptions of their roles.

### Descriptions of the src Directories

pits, modptt, and ptutor : three main directories corresponding to the three executables in the *bin* directory.

clips : directory containing the source code for the CLIPS expert system shell. CLIPS is used to implement the procedural network that some part tasks use to evaluate procedural correctness. This is the C source that implements CLIPS. The actual CLIPS code for the procedures is in the *bin* directory.

graphics : directory containing all the code used to display user interface graphics, which range from windows and buttons to plain text. The code in this directory defines such objects and manipulates them.

image : directory containing the source code used to display images on the screen, specifically, routines used to display GIF images and ipaste images (see the Silicon Graphics Programmer's Reference Manual for details on the ipaste routine).

## Makefiles

There is a makefile in each directory that will compile the source code contained in that directory. It will produce a corresponding executable program of the same name as the directory. In addition, the makefiles in the top level directories (*modptt*, *ptutor*, and *pits*) will recompile all their subdirectories if needed, and move the linked executables to the *bin* directory, where they can be executed.

Note that *modptt*, *pits*, and *clips* are written in C, whereas *ptutor* is written in C++ 2.0. This means the system must have a Silicon Graphics C++ 2.0 compiler installed when attempting to make changes and recompile *ptutor*. The C++ convention of one class per file is used, so a class name and a file name are used interchangeably in this document. However, a few files, such as *classes.h/c++* contain more than one class.

## HOW TO CHANGE THE ITS

### Overview

In this section, information on how the P2T2 was modified to support the P2T2 Intelligent Trainer is provided, as well as information on adding new commands, data, or part tasks to the system. This information is also useful for integrating the ITS with new versions of P2T2.

### Modifications to P2T2

Although modifications to P2T2 were kept at a minimum, some essential changes could not be avoided. This section brings them to your attention for the purpose of adding new commands to the P2T2/ITS interface or integrating ITS with future versions of P2T2.

The most important modification to P2T2 is the addition of `its_interface.c`, which receives and implements requests from the ITS. In ITS, the `p2t2` interface class defines the methods with which ITS can issue requests to P2T2. There are two types of requests ITS can make to P2T2: to issue commands and to retrieve data.

Commands are used by ITS to set up P2T2 for a given part task and level of complexity (see the Part Task Design Document and User Manual for more information on part tasks and levels of complexity, etc.). Commands are used to do things like load models into P2T2, set the mode, etc.

Data about the current state of P2T2 is needed by ITS to evaluate the student's performance. This is accomplished by keeping copies of P2T2 data and sending them to ITS when the values change.

See also Appendix C for more detailed information on how the P2T2 was modified.

### How P2T2 and ITS Communicate

The executable programs *modptt* (P2T2) and *ptutor* (ITS) communicate using a message queue that gets created by *pits*. All of the information needed to create and use the queue is in `src/common/common_p2t2_its.h`, which is accessed by all three programs. Once *pits* creates the queue, it starts up *modptt* and *ptutor*, then waits for them to exit before exiting itself. If one program exits but the other is still active, *pits* attempts to kill the active process, since an error of some sort has occurred. (In normal processing, both programs will exit at the same time.)

Once the programs have started, they each open the message queue created by *pits* for use in their own processes. In *modptt*, this is done in `main()` by the call to `initialize_its_queue()`, which is defined in `its_interface.c`. In turn, `initialize_its_queue()` opens the queue, and writes a "P2T2 READY" command to it for synchronization purposes. If the function cannot open the queue, then *modptt* is exited.

In *ptutor*, the queue is opened when the `p2t2` interface class is created. Since this is a global variable, the queue is opened before `main()` is executed. If the queue cannot be opened, then *ptutor* is exited. In `main()`, a call to `p2t2.synchronize()` waits for the "P2T2 READY" message to appear on the queue. Once the programs have been synchronized, either process can read from or write to the queue.

If one of the processes quits or exits, then the message queue gets closed. Both programs are set up so that if the message queue closes, they will stop what they are doing and exit gracefully. Therefore, neither program can run independently of the other.

### How to Add New Commands/Data to the P2T2/ITS Interface

To add a command to the interface, a new ID must be created so that both P2T2 and ITS can identify the command. This is done by adding a new `#define` constant in `src/common/common_p2t2_its.h`. Make sure that the value of the new ID is different from the existing ones. Command IDs are generally between 300-399.

Secondly, a new method should be added to the `p2t2` interface class that defines the command in the ITS. `p2t2_interface` is the class in which all P2T2 commands and requests for data are defined. The `p2t2_interface` class itself is defined in `src/ptutor/its/p2t2_interface.h - c++`. The new method should send the command ID and any parameters the command requires to P2T2 using a message queue provided by the `p2t2_interface` class. Do not exceed the buffer length of the queue, which is defined in `common_p2t2_its.h`.

Lastly, the command should be incorporated in `its_interface.c` in the `modptt` directory by adding it to the function `process_its_command()`. The command can either be executed within the case statement or a new function can be created. If parameters in addition to the command ID are passed, they can be read in using the `scanf()` function.

To retrieve a new piece of data from P2T2 that the ITS can read, an ID must be added to `src/common/common_p2t2_its.h` for the data you want to retrieve. Data IDs should be unique values between 0-99. Next, a new method should be added to `p2t2_interface.h-c++` that allows other classes to access the data. Use the examples of data retrieval methods already implemented. There are also examples of how to send data even if it has not changed (using the `send_data_once()` command). The important thing to remember about retrieving data is that P2T2 only sends data if the value of the data has changed.

Adding new data to P2T2 requires changes in `its_interface.c` at several places. The new type of data should be added to the structure `its_data_struct`, as well as a boolean value to tell whether it is currently active or not. `its_data_struct` keeps copies of all P2T2 data that ITS retrieves, and updates the active values once per exec loop, if needed. If the data is noisy (for example, the arm joints can change by very small amounts), then add the noise level to the `its_data_struct` also. The data will only be sent if it changes by more than the noise level.

Several functions need to be updated to the new data value. These include:

```
initialize_its_data()
send_modified_data_fields_to_its()
send_data_once()
set_data_noise()
set_active_data_value()
clear_active_data()
```

The changes are all straightforward and can be accomplished by imitating what is done for other data in these routines.

Also, it is very important to update the data fields in any functions in `its_interface.c` that change the data you are retrieving. For example, if the ITS sends a command to P2T2 to change the mode, then the function that changes the mode also needs to update the mode data field in the `its_data` struct. This is needed to properly reset the data evaluation when ITS is setting up a part task.

### **How to Integrate ITS Into a New Version of P2T2**

Integrating ITS into a new version of P2T2 requires adding the `its_interface.c` file to P2T2, and adding the other ITS modifications listed in Appendix C. Since ITS uses P2T2 variables to modify the state of the simulation, the difficulty of integrating an updated P2T2 depends on the modifications that were added. If P2T2 functions that `its_interface.c` uses are substantially changed, or P2T2 variables used by `its_interface.c` have changed for some reason, then `its_interface.c` will have to be modified to reflect those changes.

Modifications other than those in `its_interface.c` are all marked by comments `/* ITS MODIFICATION STARTS HERE */` and `/* ITS MODIFICATION ENDS HERE */` so that the extra code can be found with a simple search. The only altered files are `control.c`, `exec.c`, `main.c`, `wins2.c`, `chair.c` and `ctf.c`. All modifications to each of these files are listed in Appendix C.



## A Part Task Consists of Four Classes

The source code for a part task is distributed to different files in the *its* directory. In general, a part task consists of four classes (files):

1. a meta task class
2. a part task class
3. a part task user interface class
4. a part task evaluation class

These are described in detail below.

1. meta task class is the top level class instantiated and run by `main()`. It deals chiefly with the complexity level of the part task, setting up P2T2 to a specific state needed to run the part task at a certain level of complexity, and it keeps track of the number of trials that have been passed and failed. The meta part task is also where the editable parameters are read in (see the Part Task Design Document and User Manual). The meta part task creates both the data evaluation class and the part task class.

There is usually one meta task per part task, but if the tasks are very similar, they can share a meta task. The short fly-to tasks in the various modes (Tasks #1.2.\* and 1.3.\*; see the Part Task Design Document and User Manual) all share the same meta task, since they all have the same levels of complexity and the same part task evaluation. All meta tasks are derived from a generic class called `metaTask` whose definition can be found in `metatask.h/.c++`. Their file names have the prefix "meta".

2. part task class is used to execute the part task in real-time. It contains the loop that updates the data evaluation, the timer, and the user interface. At the end of the task it returns control back to the meta task, which decides whether another trial should be run. The part task class creates the user interface class. Part task classes are shared much more frequently between part tasks than the meta classes. For example, the visualization tasks (Tasks 3.1b and 3.2b) have different meta classes but the same part task class, and all tasks with procedures share the same part task classes. In general, there is one user interface class for each part task class. All part task class file names have the suffix "pt".
3. user interface class deals only with user interface issues. It uses the classes in the graphics directory to display information and to receive input. It decides where graphical objects should be on the screen, displays help messages, final evaluation messages, and final evaluation graphs.

Many different part tasks can use the same user interface class, since they have the same requirements for a user interface. Again, all of the tasks that deal with procedures (such as Task 4.3a Grapple), require at most five buttons with roughly the same actions attached to them, such as a real-time help button. Therefore, there is one user interface class for all the procedural part tasks, and the small differences are taken care of by input parameters to the class. All user interface class file names have the suffix "ui".

4. data evaluation class is used to evaluate the student's performance during a trial. It receives data from P2T2 about the state of the simulation, such as where the arm is, what cameras are being used, etc. It also provides the user interface with real-time hints, if needed by the student. Once the student has finished the task, it evaluates the performance and assigns a score for each of the five types of evaluations (accuracy, efficiency, safety, procedure, and camera) if they apply, and updates the student model based on the scores. (Each student has an individualized copy of the student model; these are found in their home data directories and called P2T2.IT/data/student/<name of student>/studentModel.dat.) The data evaluation class also creates a text message containing the scores and their explanations for the user interface to show the student at the end of each trial.

Like the meta task class, there is usually only one evaluation class per part task, although tasks that are very similar may use the same one. However, each part task evaluation class can use some of the many general evaluation classes available for common types of evaluations. (See Appendix A for listing of the source files/classes.) For example, many tasks use a path evaluation (which determines the student's efficiency of arm movement from point A to point B) as part of the overall evaluation, so a general path evaluation class has been created that can be used by any of the evaluation classes. All part task evaluation classes are derived from a generic class called dataEval defined in dataEval.h/.c++. All part task evaluation file names have the suffix "eval". General evaluation classes may or may not have the "eval" suffix.

### Integrating a Part Task into the System

To add a new part task, you need to create the four classes described above. If the task is similar to any of the part tasks already developed, you can probably use some of the existing classes. If not, it is best to copy the four part task classes from a task that already exists, then change the class names, and change what the methods do to implement the new task. Many of the subclasses you need may also exist, and it would be well worth your time to hunt for a class that will do what you want rather than create a new one. This would be especially helpful for classes that evaluate student performance, since there are many evaluation classes. The Part Task Design Document and User Manual gives a good overview of each of the part tasks.

After creating the part task and getting it to run by calling up the meta task from main(), the task then needs to be integrated into the tutorial and the remediation portion of ITS.

Integrating the task into the tutorial is simple. Find the enumerated type called part task type in the file common.h, which defines the order in which the part tasks are presented during tutorial mode. Simply add the part task to this list. Next, the name of the task must be added to other functions in common.c++ that return information about part tasks. These functions are:

```
getPartTask()
getPartTaskName()
getPartTaskTitle()
partTaskAvailable()
```

Note that `getPartTaskName()` returns the name of the task as it appears in the editable-parameters directory for that part task. The editable-parameters directory should also be created when creating the meta task. Again, copying and modifying an existing editable parameters file is preferred over creating one from scratch.

To use the new task for remediation during maintenance mode (see the Part Task Design Document and User Manual), add the task to the domain hierarchy node that it remediates. The domain hierarchy is a tree where each node represents a skill or a concept about the RMS (for a complete description of the domain hierarchy, see Appendix D and also the Student Model Design Document). During maintenance mode, the system tests and tutors the nodes in the domain hierarchy using the part tasks. Therefore, the part task should be added to the domain node(s) that the part task remediates.

To do this, the domain hierarchy file in the ITS *data* directory must be edited. The file is called `domain_hierarchy.dat`, and it contains the structure of the domain hierarchy. The values in `domain_hierarchy.dat` are the structure only; data for each node, such as "times-remediated" and "times-used," are kept in the student model file. (The domain hierarchy file is copied to create the student model file per student.)

Each line in `domain_hierarchy.dat` defines a domain node, its children, and the part tasks that remediate the node. The general form is:

Node decay-rate num-children num-part-tasks child1 ... childn pt1 ... ptm

An example of an actual entry in the file is:

```
ENDEFF 260000 2 1 END_EFF_RHC END_EFF_THC EndEffInt
```

where

- ENDEFF is the name of the node
- 260000 is the decay rate of the node in seconds (see the Student Model Design Document)
- 2 is the number of child nodes ENDEFF has
- 1 is the number of part tasks that can remediate this node
- END\_EFF\_RHC and END\_EFF\_THC are the children nodes, which are defined below ENDEFF in the domain hierarchy file
- EndEffInt is the part task that remediates the node. This is the same name as that returned by `getPartTaskName()` in `common.c`, and the same name as the name of the editable parameters directory for the task.

To add a part task to the node, increase the num-part-tasks field by one and add the name of the task to the end of the line. So, for the example above, to add a task called EndEffBetter to ENDEFF, the line would look like this:

```
ENDEFF 260000 2 2 END_EFF_RHC END_EFF_THC EndEffInt  
EndEffBetter
```

If EndEffInt is to be replaced with EndEffBetter, the line would look like this:

```
ENDEFF 260000 2 1 END_EFF_RHC END_EFF_THC EndEffBetter
```

Be very careful in editing this file, because if the number of children or the number of part tasks do not match the actual ones there, ITS will most likely crash at some point.

## ITS DATA FILES

### ITS DATA FILES

#### Part Task Data Directory and Editable Parameters

The directory `part_task_data` stores information for each part task. Each part task is represented by a separate directory specific to that part task, as well as by a global `editable-parameters` directory which take effect on all part tasks. The parameters in these data files significantly affect the behavior of the software, so exercise care when changing them. Refer to the see the Part Task Design Document and User Manual for details on what the parameters are and on changing the editable parameters.

#### How to Add Editable Parameters

To add extra parameters for a particular part task, open the editable parameters file in the part task's data directory and add the parameter anywhere in the file, following the format of name followed by value. It is desirable to place it next to related parameters' definitions. Be sure to explain the parameter's meaning in the file carefully, since the editable parameters files are meant to be edited by instructor (i.e., non-programmer) personnel.

Next, the parameter needs to be retrieved in the source code.

- The retrieval of global parameters takes place in the function `getGlobalEditables()` in `main.c++`.
- The retrieval of part-task-specific parameters takes place in the `setEditableParameters()` method of their meta task class, except for the whole tasks (Tasks 4.8a Deploy and 4.9a Retrieve) whose parameters are read in their evaluation classes' (`deployEval` and `retrieveEval`) `setConstants()` method.

Depending on the type of the parameter (floating point, integer, string), the parameter can be retrieved by the database class using `getFloatParameter()`, `getIntParameter()`, or `getStrParameter()`. Make sure that the name of the parameter in the code matches the name in the editable parameters file. Once the parameter is retrieved, it can be used as any other variable would be used.

## Payload Database

All numerical data regarding payloads is saved in data/payloads.dat, and all models are stored in the misc/models directory. To add a payload to ITS, the following steps should be taken:

1. Create the physical model of the payload using the Solid Surface Modeler (SSM). The payload should have the trunnions attached.
2. Create an axis model for the payload. Axis models are used in teaching the students the Payload coordinate system. The axis is attached to the payload indicating the location of the POR and the directions of X, Y, and Z. To do this, three separate models, x.bin, y.bin and z.bin in the misc/models directory should be loaded into SSM. Each axis should then be scaled so that when buried within the payload, the axis protrudes out far enough to clearly indicate the orientation of the coordinate system. The best way to do this is to load the payload and all three axis models, elongate or shorten each axis, and save with the payload removed. Do not change the attitude of the axis model, since it is properly rotated when it is loaded into P2T2.
3. Create a model of the V-Guides. Load in the V-guide model (GIDEP.bin or GIDES.bin can be used as generic V-guides), duplicate it, and arrange them at their respective positions. Save all the V-guides into one model file. In addition to the name of the V-guide file, positions of the V-guides in BACS are needed for camera usage. Use the following formula to convert BACS coordinates to model coordinates:  
$$\begin{aligned} X_{\text{model}} &= X_{\text{bacs}} + 1000 \\ Y_{\text{model}} &= -Y_{\text{bacs}} \\ Z_{\text{model}} &= -Z_{\text{bacs}} - 400 \end{aligned}$$
4. Create a model of the berth-lines. Berth lines are used in Berth and Unberth part tasks as a visual aid to help the student line up the payload with the V-guides. To build this model, load the model of a berth line or create your own in the SSM, duplicate it, and arrange them at the positions of the V-guides. Then, scale the berth lines along the Z axis so that they extent from the V-guides to the trunnions when the payload is in its low-hover position.
5. Add the data associated with the payload to the payload data file. Note that you will have to load the payload into P2T2 to get some of the information. The definition of each payload is a collection of variables, which must be listed in the correct sequence. Each variable has a name at the beginning of a line, followed by its definition. Payloads in this file do not have to be arranged in any particular order.

Comments may be added between two payload definitions, and each line of comment must be preceded by a double hash mark ("##").

The parameters in the payload data file are listed as follows. Note that for the names of model files, the extension ".bin" is assumed and therefore omitted. The format for all the position data is [x y z], and the format for all the position and attitude data is [x y z pitch yaw roll].

NAME : Name of the payload model file.

VGUIDES : Name of the V-guides model file.

GRAPPLE : Name of the grapple pin model file.

BERTHED\_POR : The position and attitude of the POR when the payload is grappled and berthed in the bay, which is in payload ID 1. This can be read from P2T2's parameter dial when the payload is berthed and grappled.

BERTHED\_EE : The position and attitude of the end effector when the payload is berthed and grappled, but in payload ID 0.

RELEASED\_POR : The position and attitude of the POR telling where the payload is released in space. The payload ID is 1.

PAY\_GRAP\_OFFSET : The offset (translation and rotation) of the payload model when it is grappled. This is found by looking at the payload's model position when grappled using P2T2's setup mode.

PAY\_BAY\_OFFSET : The offset (translation and rotation) of the payload model when it is in the orbiter bay, ungrappled. This is found by looking at the payload's model position using P2T2's setup mode.

GRAPPLE\_OFFSET : The offset (translation and rotation) of the grapple fixture model, which is a dependent model of the payload model. This can be found using setup mode and looking at the grapple fixture model's offsets.

VGUIDES\_OFFSET : Offset (translation and rotation) of the V-guides model when it is sitting in the bay. Use setup mode to find the offsets of the model.

GRAPPLE\_POS : Position of the grapple fixture on the payload. The three possibilities are TOP, STBD, and PORT.

PAYLOAD\_SIZE : Relative size of the payload, which is roughly indicated by SMALL, MEDIUM, or LARGE.

MAX\_GRAP\_TO\_EDGE : The distance, in inches, from the grapple fixture to the farthest edge of the payload. This is used by the ITSs random arm generator to ensure that the payload doesn't collide with the orbiter.

BERTHLINES : Name of the berthline model file.

AXIS\_MODEL : Name of the axis model file.

EE\_TO\_POR : X, Y and Z distance in feet describing the translation from the tip of the end effector to the payload POR.

PLCS\_MATRIX : A 3x3 matrix that defines the rotation from the tip of the end effector to the payload POR. Together with EE\_TO\_POR, the payload's coordinate system can be fully described. The format is three separate lines with each line being a row in the matrix.

JOINT\_COARSE\_RATE : The rate of each arm joint for this payload when in coarse mode. The format is [SHY SHP ELP WRP WRY WRR].

JOINT\_VERN\_RATE : The rate of each arm joint for this payload when in vernier mode. The format is [SHY SHP ELP WRP WRY WRR].

VGUIDE\_POS : Coordinates in BACS of up to four of the V-guides for the payload. They are needed for camera evaluations to make sure the student is looking at V-guides correctly. The coordinates can be obtained in the SSM with the model loaded in, and then translating those coordinates into BACS. Note the first V-guide listed should be the V-guide that is best viewed by camera A.

Once the above steps are complete, the payload becomes available to ITS, which in most cases chooses a payload randomly, often based on relative size (small, medium, or large).

### Arm Configuration Database

The arm database is contained in data/arms.dat. In this file are arm configurations that are in either a singularity or a reach limit; the file contains other related information as well. The database is used in the visualization of singularity and reach limit tasks (Tasks 3.1b and 3.2b respectively). Refer to the Part Task Design Document and User Manual for detailed information about how the parameters relate to the part tasks.

Similar to the payload data file, the definition of each arm is a collection of variables, which must be listed in the correct sequence. Each variable has a name at the beginning of a line, followed by its definition. To add an arm, just follow the format of the other arms in the file. It is desirable to copy one of the existing arm definitions and then simply change the variable values. Comments must begin with a double hash mark (##), and can only be placed between two arm definitions, not within one.



The usage of all the variables is as follows:

ARM\_TYPE: The possible choices are:

- + SHOULDER\_YAW\_REACH\_LIMIT
- + SHOULDER\_PITCH\_REACH\_LIMIT
- + ELBOW\_PITCH\_REACH\_LIMIT
- + WRIST\_PITCH\_REACH\_LIMIT
- + WRIST\_YAW\_REACH\_LIMIT
- + WRIST\_ROLL\_REACH\_LIMIT
- SHOULDER\_YAW\_REACH\_LIMIT
- SHOULDER\_PITCH\_REACH\_LIMIT
- ELBOW\_PITCH\_REACH\_LIMIT
- WRIST\_PITCH\_REACH\_LIMIT
- WRIST\_YAW\_REACH\_LIMIT
- WRIST\_ROLL\_REACH\_LIMIT
- SHOULDER\_YAW\_SINGULARITY
- WRIST\_YAW\_SINGULARITY
- PLANER\_PITCH\_SINGULARITY

LEVEL\_OF\_COMPLEXITY: (1 or 2) This provides a rough indication of the difficulty of the arm when used in the visualization tasks. Complexity level 1 arms are used in Level of Complexity (LOC) 1 and 2 in the actual task, and complexity level 2 arms are used in LOC 3 (or higher).

HC\_ACTION: A positive or negative sign followed by X, Y, Z, PITCH, YAW, or ROLL. This gives the most direct hand-controller action that drives the near singularity or reach limit arm into the singularity or reach limit.

PRE\_SINGULARITY/REACH\_LIMIT\_POR: The position and attitude of the end effector that's close to a singularity or reach limit. The hand-controller action defined above would drive the arm into it.

SINGULARITY/REACH\_LIMIT\_POR : The position and attitude of the arm that's in a singularity or reach limit.

POST\_SINGULARITY\_POR: The position and attitude of the arm which the student must drive the arm to after the above singularity is resolved. This should be defined for singular arms only and is omitted for reach limit arm configurations.

The hand-controller action provided in the file is undoubtedly one of several that can drive the arm to the desired reach limit or singularity; however, it should be the most obvious one. For LOC 2, a less obvious hand-controller action can be specified, since the student isn't penalized greatly if s/he fails to choose the correct action on the first try. This can increase the student's awareness of different ways in which the arm can be driven to a problematic configuration.

For the wrist roll reach limits, the position and attitude data doesn't suffice, since the wrist roll joint's range is far greater than one revolution (although the position and attitude of a wrist roll reach limit arm given in this file should not be apparently different from the others). It should be noted that when it's encountered, ITS explicitly sets the wrist roll joint to the appropriate reach limit.

### Domain Hierarchy Data File

See **How to Add a Part Task** for details on the domain hierarchy data file (domain Hierarchy.dat).

### Student Data Files

All the information pertaining to each student's performance is stored in the *students* directory, in which each student has a distinct data directory.

Within each student's directory, there are three different types of files. One file contains the student model (studentModel.dat), which stores the student's status in the domain hierarchy. For every part task the student has completed, there should be two files: the overall-score file ([part task name]OverallScores.dat) and the all-score file ([part task name]AllScore.dat) file.

The student's average score and best score for each LOC in the part task are saved in the overall score data file. The format is:

```
AVERAGE [total score] [efficiency score] [accuracy score] [safety score]
[procedure score] [camera usage score] [time used] [level of complexity]
[number of trials]
```

.....

```
BEST [total score] [efficiency score] [accuracy score] [safety score] [procedure
score] [camera usage score] [time used] [level of complexity] 1
```

.....

For the average score, the values are the **sums** of the scores from the completed trials. The actual average scores can be obtain by dividing all the scores by the number of trials. The scores are kept this way because it's much easier to add the scores from a new trial. Observe that for the best score, number of trials is always set to 1. Level of complexity -1 indicates the average or the best score over all levels of complexity.

The scores for every single trial the student has gone through in the part task is saved in the all-scores file. The format is:

```
[total score] [efficiency score] [accuracy score] [safety score] [procedure
score] [camera usage score] [time used] [level of complexity] ["TEST" or
"TRIAL"]
```

Note that the format is the same as for the overall score file except that the header (AVERAGE or BEST) is excluded and the number of trials, which is meaningless here, is

also excluded. "TEST" in the last parameter means the scores originate from a test (maintenance mode) and "TRIAL" means they originate from a full or part task trial.

Since the ITS software keeps track of the student's performance, it's never necessary to change the values in the data files mentioned above. However, examining these files, especially the all-scores files, might provide some insights concerning the student's performance that's otherwise lost when the scores are averaged.

The student model file (studentModel.dat) keeps a record of the student's performance by marking which skills in the domain hierarchy the student has mastered and which ones s/he needs work on. From this data, it's decided whether a test or remediation should be carried out in maintenance mode. The format is:

first line:

[current part task] [level of complexity] [task finished]

following lines:

[node name] [current state] [previous state] [time used] [last used] [times  
mastered] [last mastered] [times misused] [last misused] [times remediated]  
[last remediated] [suspicion value]

...  
...

Note that the first line is not part of the domain hierarchy, but is a convenient place to store information about where the student is in relation to the part-task tutorial sequence. It tells which part task, level of complexity, and trial within that level of complexity the student is currently working on. This is retrieved when the student logs in, so they can start where they left off when they last exited the system.

Details about the student model file and the domain hierarchy are explained in the Student Model Design Document.

### ITS misc Directory

The *misc* directory is where all the graphical data is stored, which includes different types of images as well as physical models. The content of each directory is explained below.

The GIF directory holds GIF images, which are currently used in the introduction to ITS.

The *image* directory currently only has one image (the message seen at the end of the tutoring sequence), created using Alias QuickPaint. This image is invoked using the `ipaste()` function, which ITS implements in its code. This directory should contain only non-GIF images.

All the models (orbiter, arm joints, etc.) used by P2T2 and ITS are in the *models* directory. These binary files are created using the SSM.

The *world* directory contains the image of the earth used in the performance indicator/timer for each part task.

## APPENDIX A: ITS SOURCE FILE LIST

The following files are found under P2T2.IT/source/ptutor/its:

**berthsafety** : Safety evaluation for the Berth and Unberth tasks, which is separate from the evaluation of other aspects of the task (e.g., efficiency, accuracy, etc.).

**bookkeeper** : Keeps track of levels of complexity, number of trials, and whether the trials are successful.

**cameval** : Evaluation of the student's camera usage; positions and rotates cameras to produce optimum views.

**chronos** : implements the virtual timer(vTimer) and a metronome (ticker).

**classes** : basic classes used by practically all other classes, which are coordinate, posatt (position and attitude), arm\_angles, Camera, score and box.

**clipsshell** : interface to CLIPS expert system shell.

**common** : global definitions and general functions that don't belong to any class. All P2T2 specifications are defined here.

**database** : reads the contents of the data files in the data directory, including the student model, scores files, editable parameters, payload database and arm database.

**dataeval** : definition of a generic class from which all other evaluation classes are derived.

**dmatrix** : matrix class and matrix operations. The chief functions are constructing and decoding rotational matrices.

**domain** : definitions of domain nodes and the domain hierarchy.

**grapsafety** : Safety evaluations for the Grapple and Ungrapple tasks, which are separate from the evaluation of the entire task.

**helpq** : implementation of a real-time help queue.

**ipc\_msgq** : sends and receives messages from P2T2.

**msimanager** : processes mouse events sent by qMonitor.

**p2t2\_interface** : defines all commands and data that can be sent and received from P2T2.

**patheval** : evaluation class for path efficiency and accuracy, which both pertain to flying the arm from one point to another.

**payload** : defines the payload class and payloadDatabase class, where specs for all payloads are stored.

**pl** : operations on the performance indicator

**proceval** : procedure evaluation for all procedure tasks

**procnet** : implementation of the procedure network

**ptask** : generic part task class from which other part task classes are derived.

**randarm** : random arm generator.

**segue** : displays advance organizer and segue messages.

**srarm** : database classes for singularity and reach limit arm configurations.

**student** : monitoring and keeping track of student information.

**wmanager** : ITS window manager.

**Part Tasks** : \* indicates files/classes shared by all the part tasks in the group.

**Fly-to tasks:**

metaphase - long fly-to tasks

metaghost - short fly-to tasks

ghostpt\*

ghostui\*

phaseval

ghosteval

**Visualization Tasks:**

metavsing

metavrlim

visualpt\*

visualui\*

visualeval\*

**Recognition Tasks**

metafdsing

metafdrlim

fdarmpt\*

fdarmui\*

fdarmeval\*

---

\* An asterisk (\*) indicates files/classes shared by all the part tasks in the group.

**Procedure Tasks**

metagrapple  
metaungrap  
metaberth  
metaunberth  
procpt\*  
proctaskui\*  
grapeval  
ungrapeval  
bertheval  
unbertheval  
proceval

**Whole Tasks**

metaWhole  
deployeval  
retrieval

---

\* An asterisk (\*) indicates files/classes shared by all the part tasks in the group.





## APPENDIX B: THE FORMATS OF THE DATA FILES

The following files are found under P2T2.IT/data:

### **domainHeirarchy.dat**

[node] [decay rate] [number of children] [number of part tasks] [child1 ... childn]  
[pt1 ... ptm]

### **payload.dat**

NAME : Name of the payload model file

VGUIDES : Name of the V-guides model file

GRAPPLE : Name of the grapple pin model file

BERTHED\_POR : The position and attitude of the POR when the payload is  
grappled and berthed; PLID = 1

BERTHED\_EE : The position and attitude of the end effector when the payload is  
berthed and grappled; PLID = 0

RELEASED\_POR : The position and attitude of the POR when the payload is  
released in space; PLID = 1

PAY\_GRAP\_OFFSET : The offset of the payload model when it is grappled

PAY\_BAY\_OFFSET : The offset of the payload model when it is in the orbiter bay,  
ungrappled

GRAPPLE\_OFFSET : The offset of the grapple fixture model, which is a  
dependent model of the payload model

VGUIDES\_OFFSET : The offset of the platform model when it is sitting in the bay

GRAPPLE\_POS : Position of the grapple fixture: TOP, STBD, or PORT

PAYLOAD\_SIZE : SMALL, MEDIUM, or LARGE

MAX\_GRAP\_TO\_EDGE : The distance from the grapple fixture to the farthest edge  
of the payload

BERTHLINES : Name of the berthline model file

AXIS\_MODEL : Name of the axis model file

EE\_TO\_POR : X, Y and Z distance in feet describing the translation from the tip of  
the end effector to the payload POR

PLCS\_MATRIX : 1st row of the payload's rotation matrix

PLCS\_MATRIX : 2nd row of the payload's rotation matrix

PLCS\_MATRIX : 3rd row of the payload's rotation matrix

JOINT\_COARSE\_RATE : The rate of each arm joint for this payload when in coarse mode

JOINT\_VERN\_RATE : The rate of each arm joint for this payload when in vernier mode

VGUIDE\_POS : Coordinates in BACS of up to four of the V-guides for the payload

**arms.dat**

```
ARM_TYPE [  
  +SHOULDER_YAW_REACH_LIMIT  
  +SHOULDER_PITCH_REACH_LIMIT  
  +ELBOW_PITCH_REACH_LIMIT  
  +WRIST_PITCH_REACH_LIMIT  
  +WRIST_YAW_REACH_LIMIT  
  +WRIST_ROLL_REACH_LIMIT  
  -SHOULDER_YAW_REACH_LIMIT  
  -SHOULDER_PITCH_REACH_LIMIT  
  -ELBOW_PITCH_REACH_LIMIT  
  -WRIST_PITCH_REACH_LIMIT  
  -WRIST_YAW_REACH_LIMIT  
  -WRIST_ROLL_REACH_LIMIT  
  SHOULDER_YAW_SINGULARITY  
  WRIST_YAW_SINGULARITY  
  PLANNER_PITCH_SINGULARITY ]
```

LEVEL\_OF\_COMPLEXITY : 1 or 2

HC\_ACTION : A positive or negative sign followed by X, Y, Z, PITCH, YAW, or ROLL, which gives the most direct hand-controller action that drives the following arm a singularity or reach limit.

PRE\_SINGULARITY/REACH\_LIMIT\_POR : The position and attitude of the end effector that's close to a singularity or reach limit. The hand-controller action defined above would drive the arm into it.

SINGULARITY/REACH\_LIMIT\_POR : The position and attitude of the arm that's in a singularity or reach limit.

POST\_SINGULARITY\_POR : the position and attitude of the arm which the student must drive the arm to after the above singularity is resolved. Defined only for singular arms.

**-AverageScore.dat**

AVERAGE [total score] [efficiency score] [accuracy score] [safety score]  
[procedure score] [camera usage score] [time used] [level of complexity]  
[number of trials]

.....

BEST [total score] [efficiency score] [accuracy score] [safety score] [procedure  
score] [camera usage score] [time used] [level of complexity] 1

.....

**-AllScore.dat**

[total score] [efficiency score] [accuracy score] [safety score] [procedure  
score] [camera usage score] [time used] [level of complexity] ["TEST" or  
"TRIAL"]



## APPENDIX C: MODIFICATIONS TO P2T2 SOURCE CODE

### APPENDIX C: MODIFICATIONS TO P2T2 SOURCE CODE

**main.c** : 1 modification

main():

The queue with which P2T2 communicates with ITS must be initialized:  
initialize\_its\_queue();

In case there is a left-over crosshairs:  
erase\_all\_crosshairs();

**exec.c** : 2 modifications

1. Exec() : The messages received from ITS are processed:

process\_ITS\_msg\_que();

2. otherModes() : Some flags and parameters are set as a part of the ITS test mode implementation.

**ctf.c** : 2 modifications

1. ctf() : The payload rate vectors must be initialized to 0, otherwise when the payload is grappled and after ITS sets the arm to a position, the noise in the payload rate vectors can cause the arm to start moving.

pl\_trans\_rate\_cmf[0] = pl\_trans\_rate\_cmf[1] = pl\_trans\_rate\_cmf[2] = 0.0;

pl\_rot\_rate\_cmf[0] = pl\_rot\_rate\_cmf[1] = pl\_rot\_rate\_cmf[2] = 0.0;

2. ctf() : max\_trans\_res is 0 in the absence of hand controllers, which causes a loaded arm to NOT move when the keyboard commands are being used. Therefore, the code that uses max\_trans\_res to set ee\_rot\_rate\_cmd and ee\_trans\_rate\_cmd is executed only if there is a chair installed. Note that the payload rates will be wrong in keyboard P2T2.

```
if (CHAIR) {  
    if (length > max_trans_res) {  
        .....  
        .....  
    }  
}
```

**chair2.c** - 1 modification

**ReadChair()** : There is a substantial amount of noise in the data acquired from the chair, so it is rounded off to zero if it is near zero. Currently, the modification only affects CHAIR2, which is defined as the chair received from Jim Brock in July, 1991. Other chairs can be added if they are too noisy to calibrate.

**wins2.c** : 6 modifications

1. **InitWins()** : The P2T2 input window overwrites the ITS tutor window, so its location is changed to:  
  

```
preposition(700,1200,600,650);
```
2. **DefaultDevice()** : The devices are not requeued because it changes the dials to their reset values, which changes the current mode to no mode since ITS sends changed values of the dials to be processed by P2T2.  
  

```
QueDefault()
```
3. **DoDefault()** : Modified so that the main window camera does not get changed when events other than DIAL or SW are put on the queue.
4. **UnfullWindow()** : If the crosshairs exists, it must be erased before the full window is deactivated, otherwise the crosshairs will remain in the overlay plane:  
  

```
erase_crosshair(mainid);  
RedrawMain();
```
5. **CamX(), CamY(), CamZ(), InitCamX(), InitCamY(), and InitCamZ()** : Modified to fix a bug, which is that when camera positions are changed, the old valuator for a previous movement is used to compute the current position. For example, if camera A were translated n units in the X direction and the current camera is switched to camera B, camera B is automatically translated n units in the X direction. The bug is fixed by adding relative camera positions so that cameras only move by the relative change instead of the absolute value of the valuator.
6. **ZoomOut()** : The grapple crosshairs lines up with the grapple fixture only when the wrist camera's zoom factor is exactly 1.0. The original code actually allows the zoom factor to be less than 1.0, causing the field of view to be greater than the maximum field of view of the camera. To ensure that does not occur:  
  

```
if (pCurCam->zoom * 0.9 < 1.0)  
    pCurCam->zoom = 1.0;  
else  
    pCurCam->zoom *= 0.9;
```

**control.c** - 8 modifications

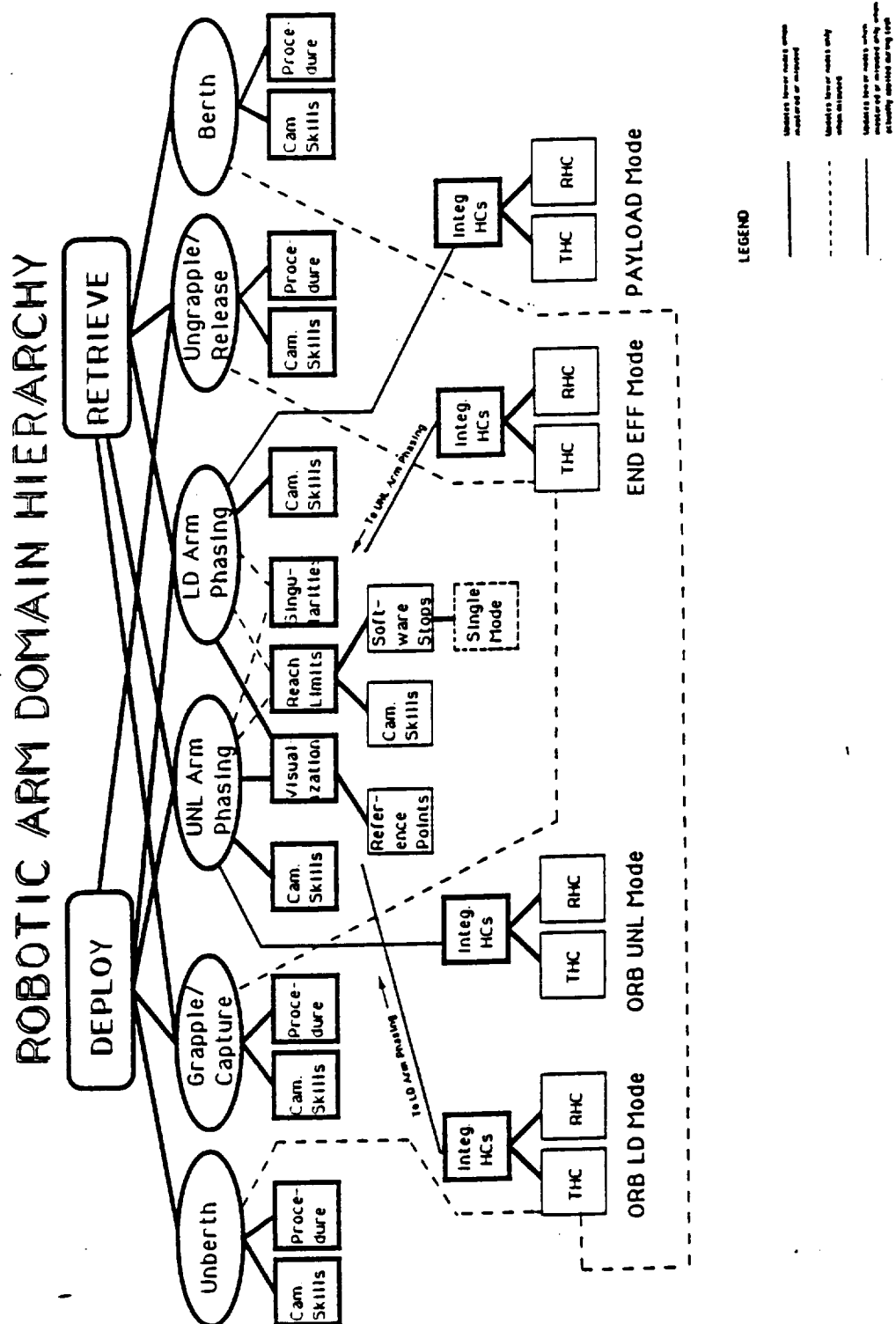
1. jointang and paramang must be non-static so ITS can access them to find what the dial positions are.
2. mciu()  
  
send\_keyboard\_input\_to\_its(val) - send all keyboard commands to ITS. This used to check for the exit command.
3. mciu()  
  
Several new keyboard commands are added.
  - "G" sets trigger\_pulled to OFF
  - "u" and "U" switch the release button ON and OFF respectively
  - "4" and "I" set the current payload ID, which are the ASCII terminal commands.
4. The crosshairs in the overlay plane is erased before exiting.
5. CheckConditions() : A mechanism that disables the audible alarm was implemented. When flag overRideAlarm is ON (which is set in its interface.c), all alarms are turned OFF. This mechanism is used in the recognition of singularities and reach limits tasks (3.1a and 3.2a) so that the alarm does not give away the answer.
6. DrawModeObj() : ITS test mode implementation.
7. Enter() : ITS test mode implementation.
8. BrakesOn() : Occasionally the arm does not stop moving after the brakes are engaged. The cause might be that the end effector translation and rotation rates are not reset to 0, which is inserted in the code:

```
ee_trans_rate_cmf[0] = 0.0;  
ee_trans_rate_cmf[1] = 0.0;  
ee_trans_rate_cmf[2] = 0.0;  
ee_rot_rate_cmf[0] = 0.0;  
ee_rot_rate_cmf[1] = 0.0;  
ee_rot_rate_cmf[2] = 0.0;
```





DOMAIN HIERARCHY FOR THE RMS





<b>APPENDIX D: SYSTEM DESIGN OVERVIEW</b>
---



domain, the input and outputs of the ITS, and to discuss the philosophical and practical benefits of several leading ITS architectures.

## 2.1 Scope of the Project

The goals of P2T2/ITS are outlined in the "P2T2 Intelligent Trainer Needs Analysis". The possible goals for the overall program include:

Development of an ITS that performs a post-session evaluation of the students performance along several parameters (e.g., safety and proficiency). This approach may allow intensive processing of student performance data to provide a detailed and optimized analysis of the student. This approach may allow for a wider range of processing options because of the potentially relaxed system response time (i.e., instructor could wait several minutes for a report to be generated). This approach could provide the following capabilities:

- identify student incorrect student actions or violations (e.g., unsafe behavior)
- provide feedback
- suggest remediation
- diagnose student errors in batch mode
  - localize student errors to a subset of all possible ones
  - isolate student errors to specific areas
  - develop or tailor simulation to help isolate/test student errors
  - confirm diagnosis through retesting
- maintain student model for each student
- compare student performance to the norm

Development of an ITS that provides training. This approach may mandate real-time response. Some of the options possible include the ability to:

- identify student incorrect student actions as they occur
- flag incorrect student actions as they occur
- stop the simulation
- backup the simulation
- provide feedback
- provide remediation

## **1. Purpose of the Design Overview**

This document gives an overview of the Prototype Part-Task Trainer Intelligent Tutoring System (P2T2/ITS). This report will serve as a "roadmap" to the more comprehensive subdesigns. The purpose of this document is to enumerate the goals of P2T2/ITS, the current constraints imposed by the domain, the high-level data flow of P2T2/ITS, and to discuss the philosophical and practical benefits of the architecture of P2T2/ITS.

The designs of the components of P2T2/ITS are fully discussed in the following documents:

- Modifications to P2T2 Design
- Performance Analysis Design
- Error Analysis Design
- Student Model Design
- Deficiencies Analysis Design
- Instructional Assignment Design

Each of these designs describes a major component of P2T2/ITS. These components are described briefly in this document.

## **2. Approach**

Our approach to developing an Intelligent Tutoring System (ITS) for P2T2 can be summarized in three steps:

Understand the problem we are trying to solve. This is done by talking to the end-user/customer and mutually agreeing to a potential approach. We are currently making these determinations.

Make a detailed study of the domain. This should include a determination of the ways things are done currently, how they should be done, and what constraints are present. We have already conducted initial interviews with instructional experts and have documented those results.

Select a technical approach which provides a solution. Also consider partitioning the solution into achievable phases so that progress can be demonstrated incrementally.

This report documents the system requirements for the development of an intelligent tutoring system (ITS) applied to the domain of the RMS and integrated with the P2T2 software. The purpose of this document is to enumerate the goals of the ITS, the current constraints imposed by the

- interactively diagnose student errors
  - localize student errors to a subset of all possible ones
  - isolate student errors to specific areas
  - develop or tailor simulation to help isolate/test student errors
  - confirm diagnosis through retesting
- maintain student model for each student
- compare student performance to the norm

### 3. Constraints

In order to develop a comprehensive design, we must first determine what is possible in the given domain. The constraints to the P2T2/I can be categorized as being either instructional, related to the existing simulator, or related to the target hardware/software of the P2T2/I. Analysis of these constraints will indicate what data is available, what type of evaluation or training is possible, and how fast the system can perform.

#### 3.1 Instructional Constraints

To discuss instructional constraints, we must first look at how the current training on the simulator is conducted and how evaluations are made. We conducted an initial investigation at NASA on Nov 14, 1981. The results indicate that the domain can be categorized along several areas:

*Domain Hierarchy* - a taxonomy of the skills and knowledge needed for the student to perform the assigned task. The domain hierarchy can be represented as a network of nodes which contain skills and knowledge elements. This structure can be used to represent or characterize the expert's skills. To date, it has not been used to represent what we have identified as "novice behavior".

*Training Objectives* - although demonstration of sufficient application of the skills and knowledge of the domain hierarchy may be one training objective, there may be others as well (e.g., proficiency or development of student's skill in self-evaluation)

*Tasks to Perform (goals)* - these may be included within the domain hierarchy as very high level goals.

*Diagnosis* - the method applied to determining the student's errors.

*Evaluation Criteria* - those factors that allow quantification of student performance.

*Remediation* - the approach to correcting the student's faulty performance.

Any system developed should provide programmatic structures to represent these types of knowledge. In the following subsections, we discuss the constraints and available knowledge that currently exists in the RMS domain.

### 3.1.1 Training Objectives

The identified training objectives are to develop student proficiency flying the RMS, developing his ability to operate it safely, and to develop his skills in self-evaluation during RMS manipulation. These objectives need to be translated into measurable parameters. Also, we should attempt to try to attach these training objectives into the domain hierarchy if possible. For example, does proficiency simply imply that the student correctly applies the knowledge and skills of the domain hierarchy or does it mean something more? If so, then this training objective is implicitly contained within the domain hierarchy. This is possible for the other objectives as well.

### 3.1.2 Tasks to Perform

The identified tasks to perform include manipulating the POR from some predetermined position to another, berthing and unberthing a payload, grappling a payload in the bay, and grappling a payload in orbit. These tasks can be incorporated into the domain hierarchy as high-level conceptual nodes (e.g., "Berthing A Payload" node) with the appropriate subordinate knowledge and skill nodes.

### 3.1.3 Diagnosis

The current diagnostic process can be summarized as:

Instructor provides feedback, but feedback diminishes as student performance increases. This method is currently not considered in ITS designs.

No quantitative information is made available or used during or after performance. Our approach may differ because it may be possible to calculate safety factors as well as show which nodes of the domain hierarchy were correctly or incorrectly applied during the simulation.

Post-performance evaluation is made infrequently. This could mean we should not provide post-performance evaluation either or we need to convince the end-users (instructors) that post-performance evaluation is made worthwhile. If post-performance evaluation is deemed undesirable, then real-time evaluation will be required. This may be



the simple form of the graphical representations suggested by the instructors or a more fully featured ITS will have interactive diagnostic abilities and perhaps even a training component (e.g., tutoring).

#### 3.1.4 Evaluation Criteria

Evaluation criteria seem to be centered around safety and accuracy. Efficiency and speed are not emphasized. Some mention was made by the instructors of "ease and smoothness" of the students' operations. This type of evaluation may require a different type of evaluation than provided by current methods using the student model as it is currently implemented.

#### 3.1.5 Remediation

Remediation is provided via feedback to the student during the simulation. Instructors have mentioned that a helpful tool would be a set of graphic depictions of the payload and RMS status in their orientation to the orbiter. They in turn, would categorize the situation and verbally explain the situation to the student. The student does see the graphic depictions. This would imply that an automated system should come to some conclusions about the orientations of the payload and RMS and generate this feedback directly to the student.

We have determined that it is possible to alter scenarios to provide detailed training in specific activities. This implies the ability to implement a system that can tailor scenarios automatically.

### 3.2 Language Constraints

The choice of language is based primarily on the availability of language compilers for the target hardware. If more than one language compiler is available, then the selection of the software will be based on issues such as available data constructs which can represent the knowledge structures thought desirable for P2T2/ITS.

Based on previous ITS research, it is most likely that entities such as schemata and objects (with their associated methods) will be prime structures for representing knowledge. We have obtained a C++ compiler for the Silicon Graphics computer. C++ is an object-oriented language based on the C programming language. C++ offers several advantages:

- Compatible with C. P2T2 is written in C.
- Has powerful object-oriented features.
- C++ libraries are available. C++ data objects and method libraries are widely available and can be used to develop P2T2/ITS.

P2T2/ITS will need some inference engine for any reasoning process that cannot be implemented in C++. The inference engine must have the following characteristics:

- The inference engine must be embedded into P2T2/ITS.
- The inference engine must be extensible to handle the unique requirements of P2T2/ITS.
- The inference engine must be fast in order to handle real-time analysis performed by P2T2/ITS.

We have chosen CLIPS as the inference engine for P2T2/ITS. CLIPS meets all the requirements listed above. CLIPS is written in C and source code is provided so it can be modified and incorporated with P2T2/ITS which will be written in C and C++.

The choice of operating system is dictated by those available for the Silicon Graphics machines. A multi-tasking operating system would allow for the simultaneous execution of the P2T2 software and the ITS (necessary for real-time evaluation and training). A single tasking operating system would limit our ITS to a post-evaluation process. In this case, the requirements for real-time evaluation graphics would have to be made by modifying the P2T2 software itself.

#### 3.2.1 Existing Simulation Software

The P2T2 simulation software dictates what types of data inputs will be available to the ITS for evaluation. We have identified that position coordinates of the arm are readily available. Also available will be the state of the various control panel switches and knobs. To access this data will require modifying the P2T2 software to output this data. This data may be shared between the P2T2 software and the ITS via common memory blocks or file sharing (i.e., the P2T2 outputs data to a file which is later read by the ITS or both).

#### 3.2.2 Performance Constraints

Performance constraints relate primarily to the allowable time for the ITS to process and present its information. To date, only one requirement has been identified. This is the requirement to present graphics in real-time. The maximum time allowed to perform a complete post-evaluation has not been determined yet.

### 4. Overview of the Architecture of P2T2/ITS

This section presents an overview of the architecture of P2T2/ITS. It represents our understanding of the needs of the end-user. We anticipate that the architecture will change as the needs change.

P2T2/ITS will be composed of six modules. The modules are:

- Modified P2T2
- Performance Analysis
- Error Analysis
- Student Model
- Deficiencies Analysis
- Instructional Assignment

These modules and the data that is passed between them are illustrated in Figure 1. Each of the modules is fully described in its own design document.

#### 4.1 Implementation Approach

The implementation of P2T2/ITS will proceed with several overall goals in mind. These goals are to enhance the adaptability of the ITS to domains and ease maintenance of P2T2 and the ITS.

**Minimize the modification of P2T2.** As P2T2 is modified and enhanced it is desirable to see those changes propagated to P2T2/ITS. If the number of changes to P2T2 required by the ITS are minimized, it will be easier to maintain the P2T2 portion of P2T2/ITS.

**Segregate P2T2 and the ITS as much as possible.** In order to make the ITS portion of P2T2/ITS adaptable to other domains and new tasks, the ITS portion and the P2T2 portion of P2T2/ITS should communicate through well-defined interfaces.

**Minimize assumptions about the RMS domain.** The ITS should make few assumptions about the RMS domain as possible. This will make the ITS more portable to other task-oriented domains. Assumptions about the RMS domain will be identified and documented.

#### 5. System Description

Figure 1 illustrates the overall functionality of the P2T2/ITS. The modules of P2T2/ITS are briefly described below. For a complete description of a module, refer to its design document.

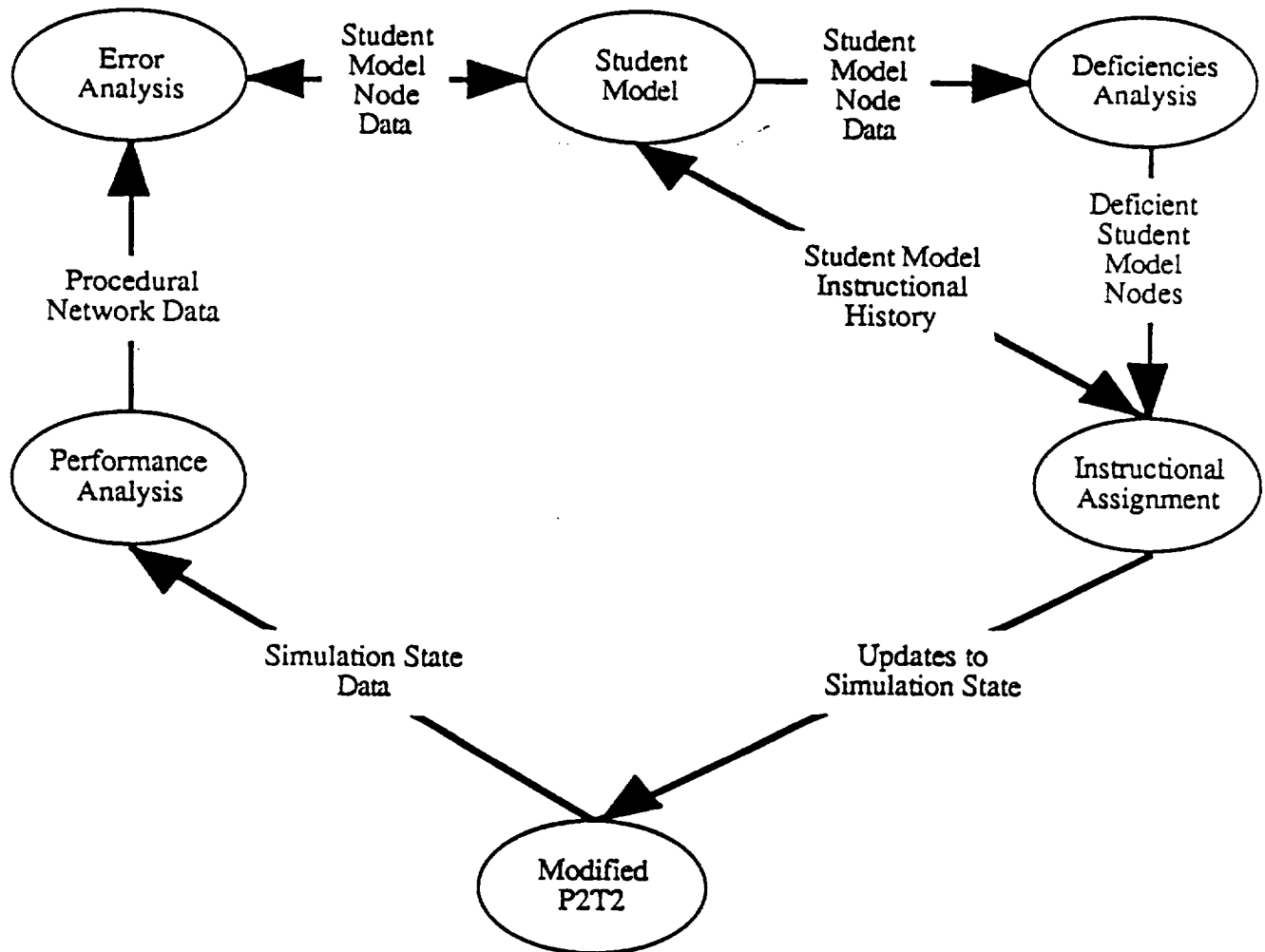


Figure 1. Overview of P2T2/ITS

## 5.1 Modified P2T2

P2T2 will be modified to pass information about the state of the R simulation as the student is performing a task to the Performance Analysis module. Information about the RMS position and rates of change, RMS mode, camera positions, and the student's action is monitored and sent.

## 5.2 Performance Analysis

Performance Analysis performs an initial analysis of the student's manipulation of the RMS. Performance Analysis only analyzes the student's current performance and does not look for possible historical causes of the student's behavior. The historical analysis of the student is left to Error Analysis and the Student Model modules.

To do this, Performance Analysis must contain use a representation expert performance on which to base its analysis of the student. Performance Analysis just analyzes the student's current performance. It does not attempt to explain the student's mistakes or his or her progress over time.

Performance Analysis must analyze the student over two different sets of criterion. Performance Analysis must analyze how well the student performed a procedure or task in the RMS domain and how well the student followed safety, efficiency, and accuracy constraints while performing the task or procedure. For example, Performance Analysis must determine how well the student performed a procedure such as grasp a payload in orbit, berthing it in the cargo bay, and stowing the air. It must also check if the student performed the correct sequence of actions. Global constraints such as safety, efficiency, and accuracy are monitored over the entire task and cannot be determined just by looking at one step in the procedure or task.

### 5.2.1 Task Analysis by a "procedural network"

We have chosen to represent the procedural analysis that Performance Analysis must do with a "procedural network"<sup>1</sup>. A procedural network is a powerful representation for interpreting a student's actions as he or she is performing the procedure or task. Briefly, the advantages of a procedural network are:

- can be used for real time evaluation of the procedure
- multiple levels of abstraction in the procedure
- mechanisms for representing the partial ordering of procedures

---

<sup>1</sup>Sacerdoti, D. (1977). *A Structure for Plans and Behavior*. Elsevier North-Holland, New York.

- a representation of the "world" as it relates to the procedure

Procedural networks have a venerable history in intelligent tutoring systems. Many researchers have chosen procedural networks as a representation of the knowledge domain.<sup>2,3,4</sup> In Phase I of the project we found the procedural network is a powerful representation of task oriented domains.<sup>5</sup>

#### 5.2.2 Constraint Analysis

While the procedural network provides a step-by-step analysis of a student's performance, constraints like safety, efficiency, and accuracy must be measured over the whole task or procedure. A constraint like "safety" could be measured monitoring how close all parts of the RMS are to the payload and shuttle. "Efficiency" might be measured by often the student uses the both hand controllers simultaneously or how much time the student spends getting out of singularities. We are defining how these global constraints can be measured.

#### 5.3 Error Analysis

Once Performance Analysis has determined what mistakes the student made while performing a task, Error Analysis must determine why the student made mistakes and pass those explanations to the Student Model. The procedural network lends itself to a convenient classification of student errors.<sup>6</sup>

#### 5.4 Student Model

The Student Model is represented by a "domain hierarchy". The domain hierarchy is a hierarchical network representing the skills and knowledge needed for the procedure. The skills and knowledge are decomposed into subskills and subknowledge down to "atomic" skills and knowledge of the RMS domain. This decomposition of the RMS domain must take the information it receives from Error Analysis and incorporate it into its historical model of the student.

The Student Model contains two kinds of historical information about the student, performance history and instructional history. Performance history is obtained by the Performance Analysis and Error Analysis

---

<sup>2</sup>VanLehn, K.; and Brown J. S. (1980). Planning Nets: a representation for formalizing analogies and semantic models of procedural skills. In Snow, R.; Frederico, P.; and Montague, W. (Eds.) *Aptitude, Learning and Instruction: Cognitive Process Analyses*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

<sup>3</sup>Rickel, J. (1988). An Intelligent Tutoring Framework for Task-Oriented Domains. *Intelligent Tutoring Systems June 1 - 3, 1988*, Montreal, pp.109-115.

<sup>4</sup>Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers, Los Altos, California.

<sup>5</sup>NASA Phase I SBIR final report on the Intelligent Procedure Evaluator

<sup>6</sup>Rickel, J. *ibid*.

modules. Instructional history is passed to the Student Model from the Instructional Assignment module.

#### 5.5 Deficiencies Analysis

Since the domain hierarchy is represented as a hierarchical network, it can use algorithms drawn from electronic circuit fault diagnosis to determine information about the node or region in the Student Model that is the probable source of the student error. We have used these algorithms in an ITS developed for the United States Army in a Phase II SBIR grant.

Deficiencies Analysis must locate a specific area of the Student Model that is the source of the student's misconceptions or difficulties. It attempts to find the simplest explanation for the deficiencies it observes in the Student Model. The result is a set of skills and concepts that will be tutored by the Instructional Assignment module.

#### 5.6 Instruction Assignment

Instructional Assignment must determine how to remediate the set of deficient skills and concepts found by Deficiencies Analysis. It uses instructional history stored in the Student Model to determine how the student learns best. Instructional Assignment selects a set of "part-tasks" designed to teach a specific skill or concept. The payloads, camera settings, and RMS position are sent to the modified P2T2.





**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix E: System Architecture Design Document**

---

<b>APPENDIX E: SYSTEM ARCHITECTURE DESIGN DOCUMENT</b>
--



Section	Section Description	Page
1.	Introduction.....	3
2.	System Modes .....	4
2.1	Overview of Simulation Mode .....	4
2.2	Overview of Tutorial Mode.....	4
2.3	Overview of Maintenance Mode .....	4
2.4	Mode Selection: Simulation or Tutorial/Maintenance Mode .....	4
2.5	Mode Selection: Tutorial or Maintenance Mode .....	5
3.	Tutorial Mode .....	7
3.1	Completion and Control of the Part Task Sequence .....	7
4.	Maintenance mode .....	9
4.1	The Maintenance Loop .....	9
4.2	The Test or Tutor Decision .....	11
4.2.1	Testing Part Tasks .....	11
4.2.2	Tutoring Part Tasks .....	11
4.3	Selecting a Test: Fault Isolation Algorithms .....	12
4.4	Analyzing Student Performance in a Testing Part Task .....	12
4.4.1	Procedural Analysis and Evaluation .....	12
4.4.2	Safety Evaluation.....	12
4.4.3	Efficiency Evaluation .....	12
4.4.4	Camera Skills Evaluation.....	12
4.5	Results of Performance Analysis: Node Propagation .....	13
4.6	Instructional Assignment.....	13

**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**  
**NASA P2T2 Intelligent Trainer Design**  
**System Architecture Document**  
**February 7, 1991**

**List of Figures**

Figure	Figure Description	Page
1	System Modes .....	6
2	The Part Task Sequence .....	8
3	The Maintenance Loop .....	10

## **1. INTRODUCTION**

This document describes the system architecture of the Prototype Part-Task Trainer Intelligent Trainer (P2T2/IT). It describes the different operating modes, how they are chosen, and how the student can interact with them. It also describes how the major components of P2T2/IT work together. The major components of P2T2/IT are:

- Performance Analysis
- Error Analysis
- Student Model
- Deficiencies Analysis
- Instructional Assignment

These components are fully described in their own documents:

- Design Overview
- Modifications to P2T2 Design
- Performance Analysis Design
- Error Analysis Design
- Student Model Design
- Deficiencies Analysis Design
- Instructional Assignment Design

## 2. SYSTEM MODES

P2T2/IT has three modes of operation: simulation mode, tutorial mode, and skill-maintenance mode. Each of these modes is designed to provide the student with various training benefits. In simulation mode, the student uses P2T2 in a "free-play" format without any instructional guidance or intervention. Tutorial mode takes the student through a carefully-determined sequence of part tasks. Maintenance mode is a "loop" of tests selected by the Intelligent Trainer in order to diagnose student weaknesses, with part tasks (or sections of part tasks) then presented to remediate any student mistakes that are detected.

It should be emphasized that the P2T2/IT is a *mastery* trainer. That is, for both Tutorial Mode and Skill Maintenance Mode, the student must master the domain to the degree indicated in the various performance criteria (see the Part Task Design Document). Since computers are infinitely patient and non-judgemental, it will handle slow students quite well. In addition, various aspects of the P2T2/IT are optimized for advanced students as well; for example, if allowed by instructors, a student who succeeds on the first two trials of any level of complexity has "proficiencied" that level, and proceeds to the next.

### 2.1 Overview of Simulation Mode

Simulation mode allows a student to practice with P2T2 without any instructional guidance or intervention. No instructional feedback (visual aids, performance indicator, hints, etc.) is given to the student. No instructional control (levels of complexity, trials, performance evaluation, record keeping, etc.) is exerted over the student. The student can choose a payload and initial configuration of the RMS.

### 2.2 Overview of Tutorial Mode

Tutorial mode presents an ordered sequence of part tasks created to lead to successful whole task performance. This mode is designed to give first-time students the basic skills and knowledge of the RMS domain. The student must progress through all levels of complexity for a part task before proceeding to the next part task. Instructional feedback is available to the student (visual aids, performance indicator, hints, and evaluation feedback). All aspects of Tutorial Mode are fully described in the Part Task Design Document.

### 2.3 Overview of Maintenance Mode

Maintenance mode is designed to selectively test and train a student's skills and knowledge to maintain proficiency in the RMS domain. This mode assumes a student has completed the Tutorial Mode, and thus has, at one time, mastered the entire domain as represented by this trainer. Skill maintenance is performed using a "maintenance loop." Tests are selected by the Intelligent Trainer to discover and isolate specific areas of student misconceptions. Once these have been determined, the Intelligent Trainer selects a part task to remediate the misconceptions.

#### **2.4 Mode Selection: Simulation or Tutorial/Maintenance Mode**

Given the three possible modes of P2T2/IT, some choice must be made among them. This is illustrated in Figure 1. A student signing on to P2T2/IT is given a choice between the simulation mode and training (whether tutorial or maintenance mode). If the student selects training, the Intelligent Trainer makes the selection of tutorial or maintenance mode.

#### **2.5 Mode Selection: Tutorial or Maintenance Mode**

When the student selects training, the Intelligent Trainer must select tutorial or maintenance. For a first-time student, the Intelligent Trainer starts the student in the part task sequence. If the student has not completed the part task sequence, they are allowed to resume the part task sequence at the point where they stopped. If the student has completed the part task sequence, the Intelligent Trainer selects maintenance mode and starts the maintenance loop.

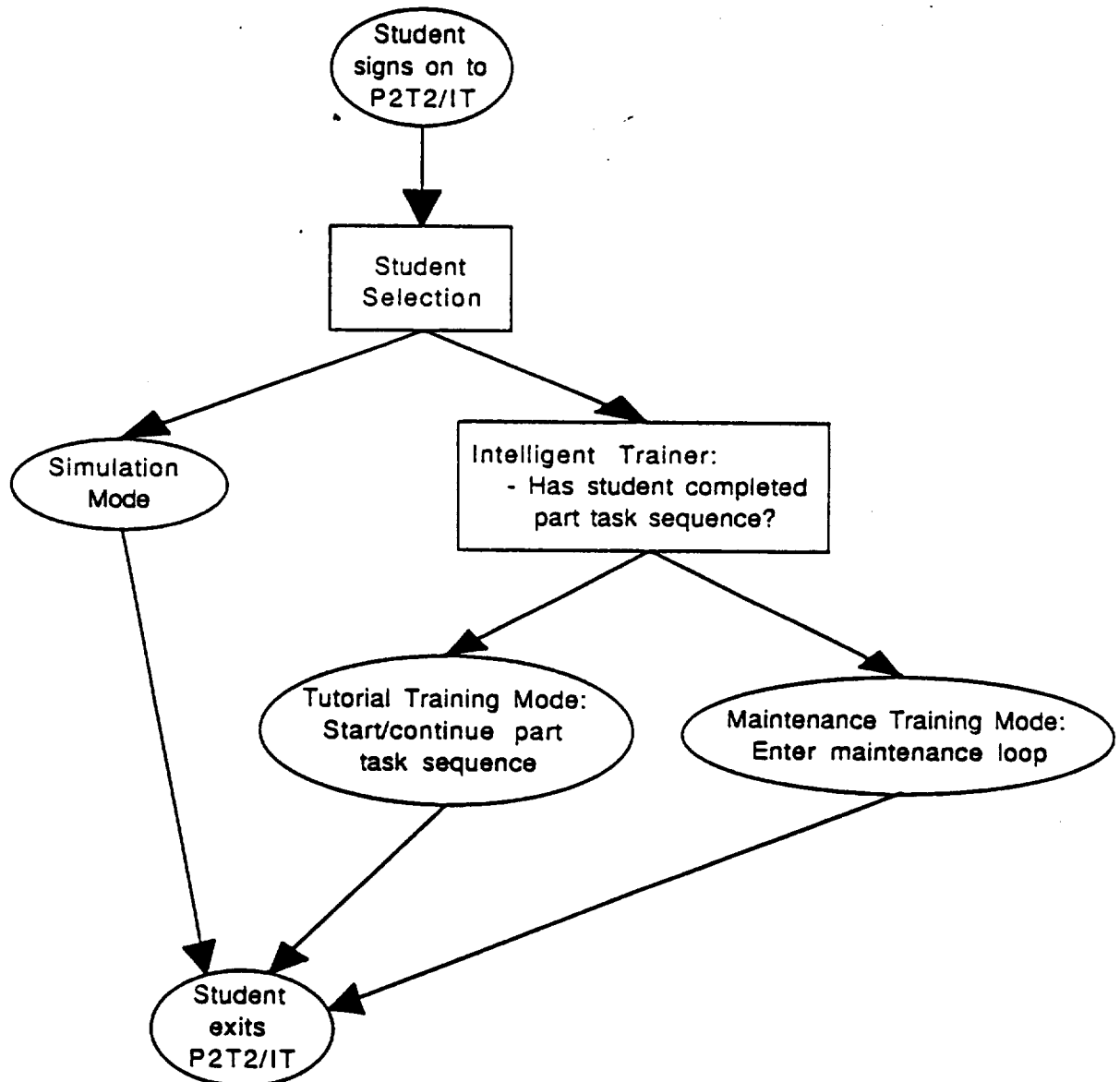


Figure 1: System Modes



### **3. Tutorial Mode**

Tutorial mode leads the student through an ordered sequence of part tasks. This sequence is designed to build up the student's proficiency from basic skills and knowledge to more complex skills and concepts. The sequence of part tasks and the part tasks themselves are determined from an analysis of the RMS domain in light of several tested instructional theories (see the Part Task Design Document). A skill or concept in the RMS domain is broken down into its constituent subskills and subconcepts. Part tasks are designed to teach one particular skill or concept in isolation; or to integrate one or more previously-taught skills or concepts.

Thus part tasks are developed to teach the skills and knowledge that make up the RMS domain. The part tasks are ordered into a carefully-constructed sequence. Part tasks that train the subskills of a particular skill are taught and mastered before the skill itself is taught by a part task. See Figure 2 for an illustration of this.

#### **3.1 Completion and Control of the Part Task Sequence**

To complete the part task sequence, the student must complete each part task in order until the sequence is exhausted. To complete a part task, the student must complete all the levels of complexity and perform within the evaluation criteria set by editable parameters.



$S_1$  and  $S_2$  are subskills of  $S$

$S_{11}$ ,  $S_{12}$ ,  $S_{13}$  are subskills of  $S_1$

$S_{21}$  and  $S_{22}$  are subskills of  $S_2$

Figure 2: The Part Task Sequence

#### **4. Maintenance Mode**

Maintenance mode is designed to maintain a student's skills and knowledge of the RMS domain by testing and tutoring selected tasks in the domain. This section describes the "maintenance loop" of testing and tutoring. The algorithms that drive the maintenance loop are designed to isolate specific student misconceptions efficiently, and to choose the part task(s) best suited to remediate them.

##### **4.1 The Maintenance Loop**

The maintenance loop is illustrated in Figure 3. It employs all the functional modules of the Intelligent Trainer: Error Analysis, Performance Analysis, the Student Model, Deficiencies Analysis, and Instructional Assignment. All these modules are described in detail in their own design document. This section will describe how they interact in the maintenance loop. The maintenance loop assigns "testing" part tasks to further refine the Student Model's record of what the student knows or does not know. The maintenance loop assigns "tutoring" part tasks once a specific misconception has been isolated by testing.

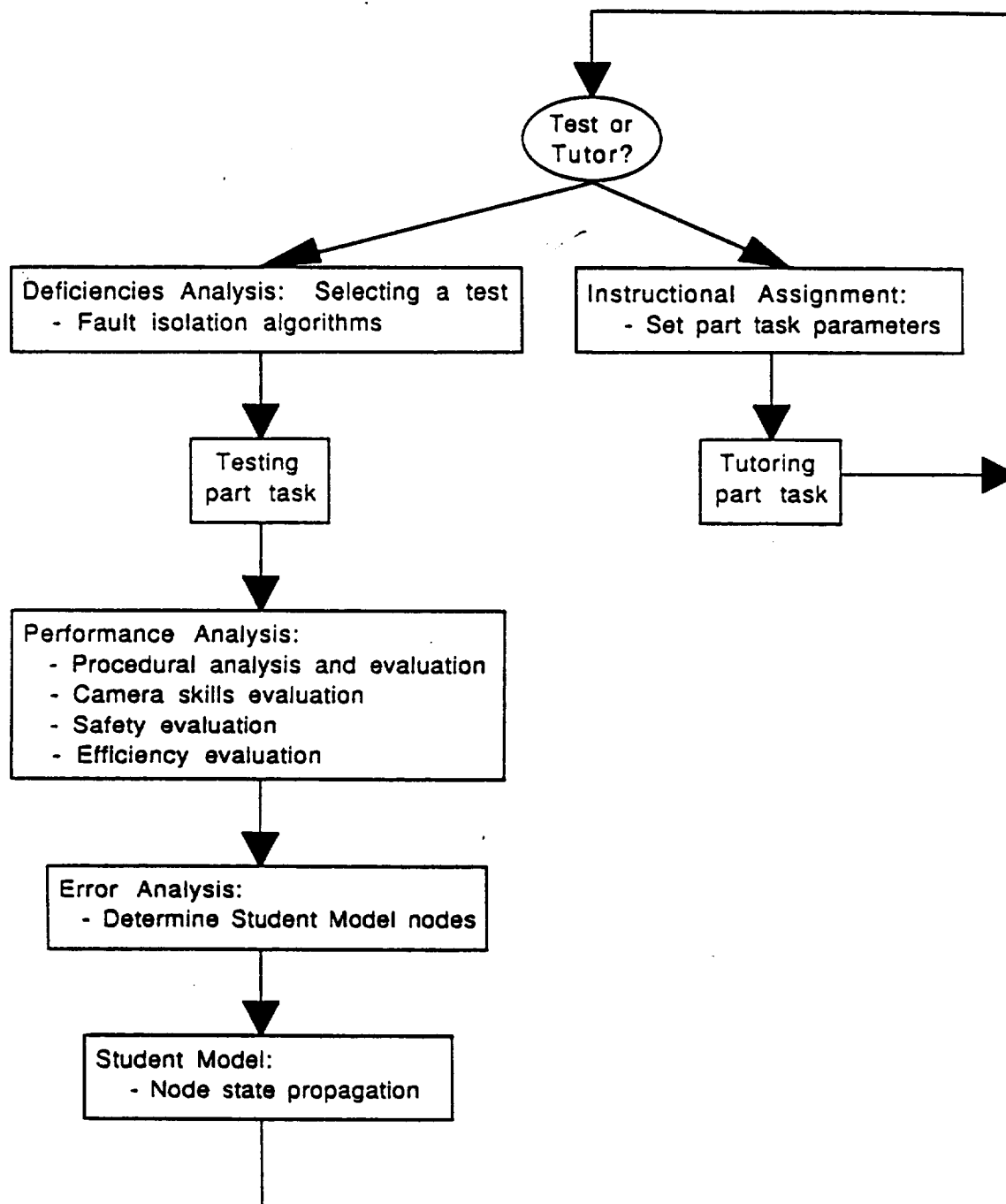


Figure 3. The Maintenance Loop

#### **4.2 The Test-or-Tutor Decision**

When the student enters the maintenance loop, the Intelligent Trainer must select a test to further refine the source of the student's misconceptions, or select a part task to tutor a misconception that has been isolated. This decision is based on the current state of the Student Model. The Student Model contains a hierarchical decomposition of the skills and knowledge of the RMS domain.

The Student Model contains nodes representing the skills and concepts of the RMS domains. Some of these nodes may be marked "mastered," indicating that the student has demonstrated mastery of the concept or skill represented by the node. Conversely, nodes may be marked "misused," indicating that the student has actually demonstrated some fault in his/her understanding of the skill or concept. The misconception may be with that particular skill or concept or it may be a symptom of a misconception in a supporting subskill or subconcept. The maintenance loop is designed to discriminate efficiently between symptoms and the actual source of the student misconceptions.

The test-or-tutor decision is based on the state of the Student Model. If the Student Model shows that a node has been marked as the source of a student misconception, then that skill is selected for tutoring by a part task. If no node has been marked as the source of a student misconception, then a further test must be performed. The Student Model marks a node as the source of student misconception if the node is known to have been misused and all its supporting subskills and subconcepts are known to have been mastered, or, if the node is misused and it has no supporting subskills and subconcepts. See the Student Model design document for a complete description of its functionality.

##### **4.2.1 Testing Part Tasks**

Testing part tasks are designed to test one skill or concept in the RMS domain. They are based on the tutoring part tasks but do not have the instructional feedback (visual aids, hint feedback, performance indicator, levels of complexity). Each node in the Student Model has a testing part task associated with it that will test the skill or concept that the node represents.

##### **4.2.2 Tutoring Part Tasks**

The tutoring part tasks used in maintenance loop are the same individual part tasks used in the initial tutoring sequence. They have the same evaluation parameters, levels of complexity, and instructional feedback. Each node in the Student Model has a tutoring part task associated with it that will tutoring the skill or concept that the node represents. The algorithms to determine which level of complexity to begin tutoring the student after a diagnosis has been made are yet to be determined.

#### **4.3 Selecting a Test: Fault Isolation Algorithms**

If the Student Model has not isolated a specific misconception, the Deficiencies Analysis module must select a node in the Student Model to test. This is done using the fault isolation algorithms of the Deficiencies Analysis. They are designed to select the most efficient test that will reveal the most information about the student's mastery or misconceptions of the RMS domain. Once a node in the Student Model has been selected, its associated testing part task is performed.

#### **4.4 Analyzing Student Performance in a Testing Part Task**

As the student is performing a testing part task, the Intelligent Trainer is evaluating his or her performance. Performance Analysis must determine what mistakes the student made and Error Analysis must determine which skills and concepts in the Student Model must be updated. Performance Analysis is done on a number of criteria: procedural performance, camera skills, safety, and efficiency.

All the different criteria that Performance and Error Analysis use to make their analyses must relate back to the skills and concepts recorded in the Student Model. The result of Performance and Error Analysis is a set of skills and concepts in the Student Model that were mastered or misused by the student.

##### **4.4.1 Procedural Analysis and Evaluation**

Procedural Analysis is done by a "procedural network." Student actions are monitored by the procedural network as he or she performs a part task. When a student makes a mistake, Performance Analysis locates the step in the procedure where the error occurred. Each step in the procedure is mapped to a node in the Student Model that represents the skills and concepts to perform the step. \*\*\*\*

##### **4.4.2 Safety Evaluation**

Safety is monitored by checking for collisions between the RMS, the Shuttle, or payload.

##### **4.4.3 Efficiency Evaluation**

Efficiency is evaluated by monitoring the time the student takes to perform a task (when time is part of the evaluation criteria, and when the parameter to use time as a measure of efficiency is active), how many reach limits or singularities the student encounters, and the path of the arm.

##### **4.4.4 Camera Skills Evaluation**

The student's use of cameras is monitored. The evaluation determines if the student is using the appropriate camera and if the best angle is chosen.

Section	Section Description	Page
1.	INTRODUCTION .....	4
2.	SYSTEM ARCHITECTURE OF P2T2 .....	5
2.1	The Structure of P2T2.....	5
2.2	Part-Task Training .....	8
2.2.1	Part-task Training Requirements .....	8
3.	IMPLEMENTATION OPTIONS.....	9
3.1	Approach 1 - Embed IT into P2T2.....	9
3.1.1	Modifications to P2T2 .....	9
3.1.1.1	Advantages .....	9
3.1.1.2	Disadvantages.....	9
3.2	Approach 2. IT and P2T2 as separate processes .....	12
3.2.1	Advantages .....	12
3.2.2	Disadvantages.....	13
3.3	Conclusions .....	13
4.	INTERPROCESS COMMUNICATION BETWEEN P2T2 AND THE INTELLIGENT TRAINER.....	15
4.1	Sending and Receiving Data.....	15
4.2	Sending and Receiving Commands.....	16
4.3	Sending User Actions.....	17
5.	IMPLEMENTATION GUIDELINES.....	18

<b>Figure</b>	<b>Figure Description</b>	<b>Page</b>
1.	P2T2's event loop .....	5
2.	Overview of P2T2.....	6
3.	Overview of IT embedded into P2T2.....	10
4.	Overview of IT and P2T2 as separate processes .....	13



## **1. INTRODUCTION**

This document describes the modifications of the Prototype Part-Task Trainer (P2T2) for the NASA P2T2 Intelligent Trainer (P2T2/IT) and other implementation-related topics. For an introduction to the functions of the other modules, refer to the following documents describing other modules and aspects of the P2T2/IT:

- P2T2 Intelligent Trainer Design Overview
- P2T2 Intelligent Trainer Performance Analysis Design
- P2T2 Intelligent Trainer Error Analysis Design
- P2T2 Intelligent Trainer Student Model Design
- P2T2 Intelligent Instructional Assignment Design

## 2. SYSTEM ARCHITECTURE OF THE P2T2

In order to simulate the Remote Maneuvering System (RMS), P2T2 must simulate the three-dimensional objects of the Shuttle, RMS, and payloads as well as the functionality of the Master Control Interface Unit (MCIU) and the General Purpose Computer (GPC). The physical objects that P2T2 simulates (the Shuttle, the RMS, payloads) are three-dimensional descriptions called "object models." They can be scaled, rotated, transformed, and viewed from different perspectives. How the object models are manipulated is determined by a "transformation matrix" maintained by the Silicon Graphics graphics library.

P2T2 must determine how the simulation should affect the display of the object models. P2T2 takes input from various sources (RHC\*, THC, keyboard, dials, buttons, and mouse) and determines how that input should affect the displayed position of the RMS. For example, pressing a button might turn on a light on the simulated RMS control panel, or moving the RHC could change the position of the RMS. Figure 1 illustrates how P2T2 reads input from various sources and changes the simulation display.

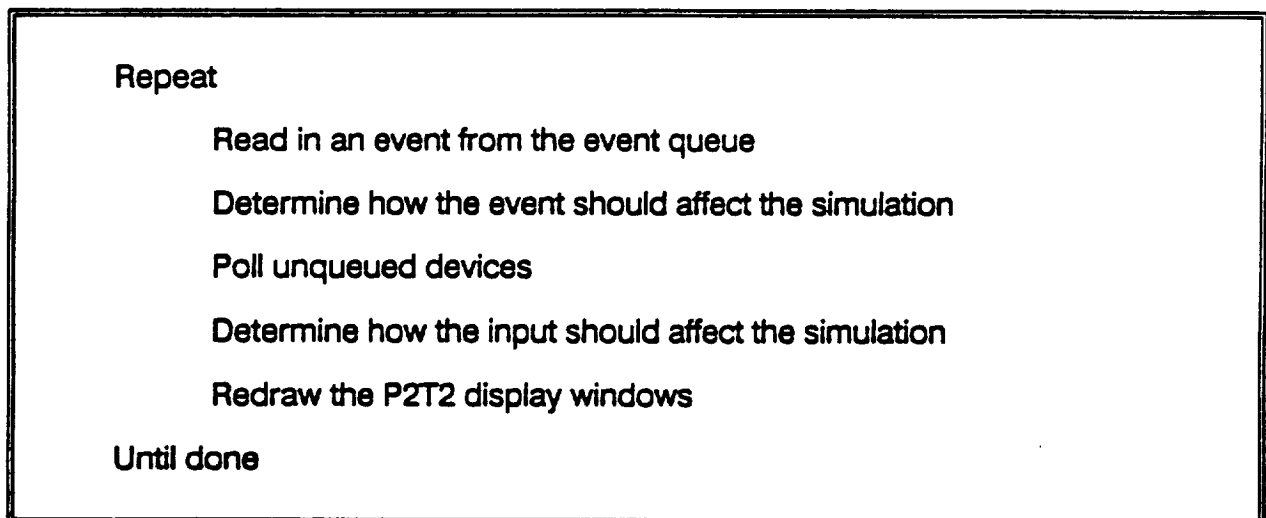


Figure 1. P2T2's event loop

### 2.1 The Structure of P2T2

P2T2 is driven by an "event loop." In one iteration of the event loop, input is read in (from the hand controllers, keyboard, buttons and dials, and the mouse), the state of the simulation is changed (RMS modes are changed, camera views changed, the RMS moved), and the windows are redrawn to reflect the change to the simulation. The event loop is the main control structure in P2T2.

P2T2 has a hybrid event loop. Some input devices are read from an "event queue." The event queue is a "first in, first out" (FIFO) queue maintained by the Silicon Graphics

\* RHC = Rotational Hand Controller  
THC = Translational Hand Controller

graphics library. When a queued input device is changed (for example, pushing a button or clicking the mouse in a window) an "event" is generated and placed in the event queue. The event contains information about the device that was changed as well as how it was changed. The keyboard, mouse, dials, and buttons place their input in the event queue. Events are held in the queue until they are read by P2T2, which then decides how they should affect the simulation. For example, an event that records a buttonpress may in fact change the camera view in the current window, or a keypress may change the RMS's movement rate from "coarse" to "vernier."

Not all the input devices generate events in the event queue. The RHC and the THC are polled input devices: they must be periodically "polled" to determine how they have been changed. P2T2 must regularly check the state of the device or "poll" it to see if it has been changed. Figure 2 illustrates P2T2's event loop and how P2T2 handles polled and queued input devices.

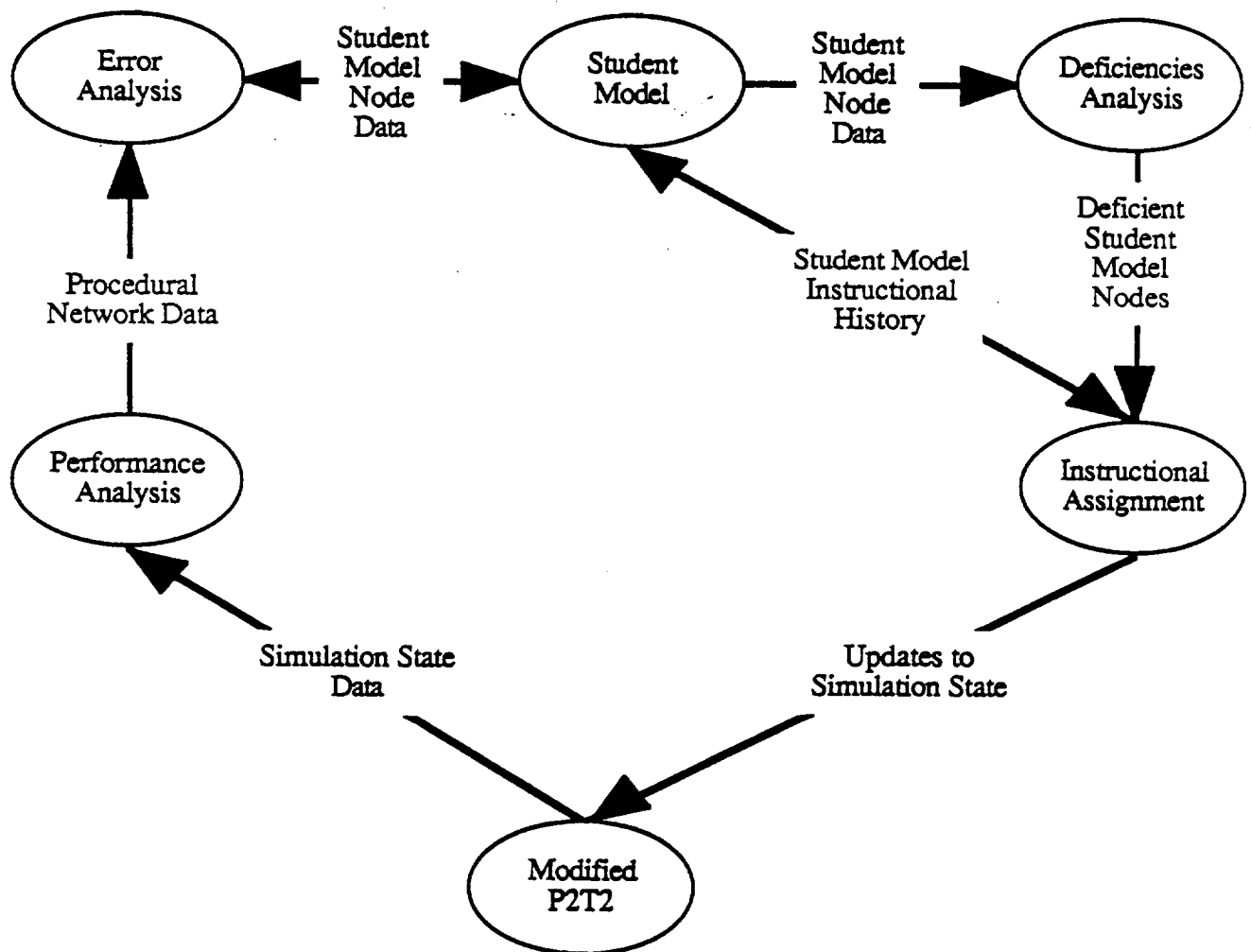


Figure 2. Overview of P2T2

## **2.2 Part-Task Training**

Proper use of a simulation may require complex skills and concepts. These skills and concepts may be difficult to develop individually in the simulation. In our approach to training, the complexity of the simulation must sometimes be reduced in order to develop basic skills and concepts or to develop an individual skill or concept.

We have decided to provide remediation with part-task training (see the "Part Task Design Document"). A part task seeks to develop an individual skill in isolation. To do this, a part task may use a situation that is not strictly "realistic," but which is designed to develop the specific skill. For example, a part task might use a visual aid like crosshairs while training grappling a payload to help the student develop proficiency in grappling.

### **2.2.1 Part-task Training Requirements**

Part-task training will require modifications of P2T2. Some part-tasks will need some capability that must be added to P2T2. The following list is a summary of the modifications and extensions to P2T2 required for part-task training. (See the "Part Task Design Document" for a complete explanation of each part task.)

- Load new object models into the simulation (such as visual aids)
- Display and hide new object models
- Load a new simulation scenario (payload, RMS configuration, visual aids, and initial simulation state)
- Pause a simulation scenario to provide instructional feedback
- Stop a simulation scenario
- Reset a simulation scenario to some previous point
- Alter the simulation controls (disable the RHC, THC, keyboard, buttons, dials, and mouse)
- Provide simulation state data (RMS position and attitude, mode and rate settings, position and attitude of grapple fixtures, payloads and other objects, etc.) to the IT

### 3. IMPLEMENTATION OPTIONS

#### 3.1 Approach #1 - Embed IT into P2T2

One way to integrate the IT and P2T2 is to embed the IT into P2T2. Both the IT and P2T2 would then run as one process. This approach is illustrated in Figure 3. Any visual aids used by the IT would be added to the object models of the Shuttle, RMS, and payloads. An IT "executive" would intervene between the RMS simulation modules and the transformation matrix. The IT executive would control the simulation scenario as well as the simulation controls.

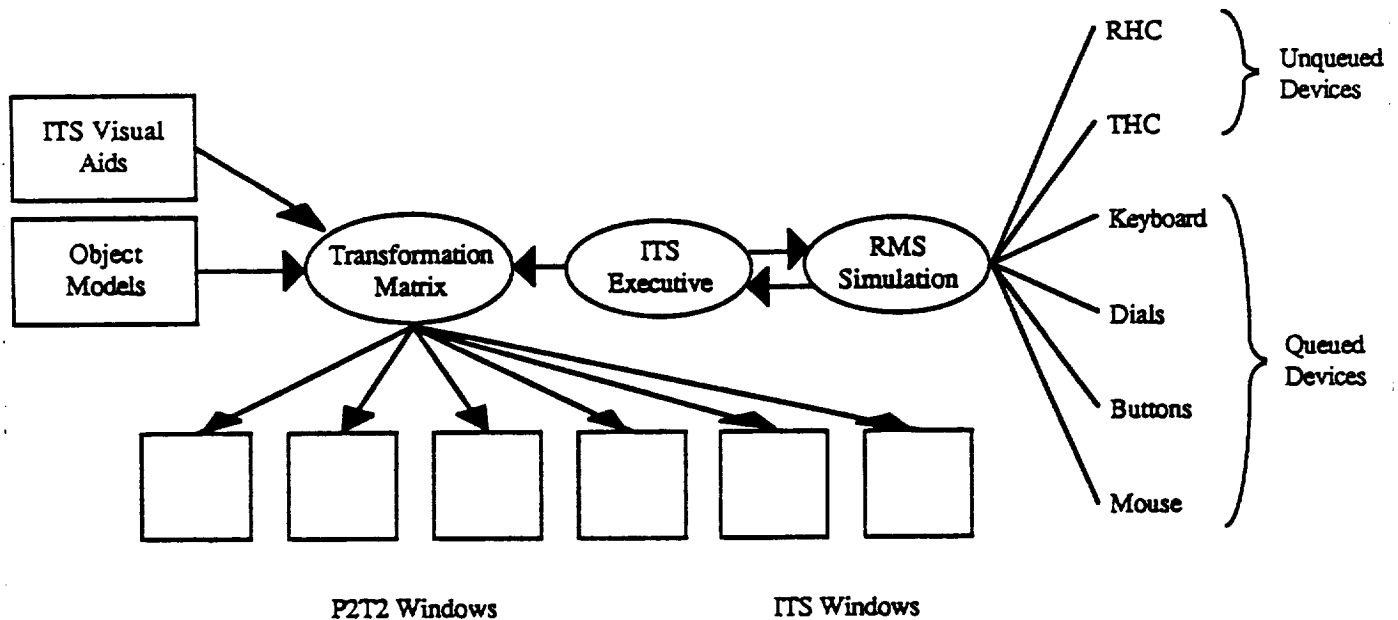


Figure 3. Overview of IT embedded into P2T2

### 3.1.1 Modifications to P2T2

- Restructure high level control of P2T2 to carry out part-task training under the control of the IT executive:
  - Load new models
  - Control display of models
  - Change simulation controls (RHC, THC, etc.)
  - Start new simulation scenario
  - Halt simulation scenario
  - Reset simulation scenario
- Monitor student performance. The RMS simulation modules that process input must provide input to the IT executive on student performance.
- Monitor simulation state. The RMS simulation modules that maintain the simulation must provide state information to the IT executive.

#### 3.1.1.1 Advantages

- Extensive control of P2T2. The IT executive should give us extensive control of the simulation.
- Flexibility in part-task training. Given the extensive control of P2T2, there will be great flexibility in how part tasks are designed and implemented.

#### 3.1.1.2 Disadvantages

- IT is less portable. If the IT were embedded directly into P2T2, it may be less easily ported to other simulations and systems in future development. Embedding the IT into P2T2 allows us to make assumptions about the hardware and software architecture that might not hold in other simulations. To some extent, this can be avoided by careful implementation.
- Controlling the P2T2 source code. Extensive modification of the P2T2 source code will create maintenance problems between P2T2 and P2T2/IT. To some extent, this can be avoided by careful implementation and control of the source code.
- Extensive modification of P2T2 is time-consuming. Extensive modifications of P2T2 require intimate knowledge of P2T2 that take time to acquire.

### 3.2 Approach #2. IT and P2T2 as separate processes

The second possible approach to integrating the IT and P2T2 exploits UNIX's multitasking capabilities. The IT and P2T2 would be developed as independent programs run as separate processes. P2T2 would send the IT data on the student's performance and the state of the simulation. The IT sends P2T2 commands to control the object models to display or hide visual aids, and commands to control the simulation.

The communication between processes could be implemented several ways. Each of the following techniques have advantages and disadvantages:

- Shared memory
- Pipes
- Streams

Shared memory would probably provide the fastest means of communication between the IT and P2T2. However, shared memory will not work over a network. This would prevent the IT from running on a separate machine. Pipes and streams would probably be slower than shared memory but could operate over a network. Streams are designed to handle interprocess communication and would be more flexible and faster than pipes. In any event, the link between P2T2 and the IT should be designed as a "black box" that does not depend on a particular means of communication. A black box interface would limit the changes to the IT and P2T2 if the IT were ported to a different platform.

### **3.2.1 Advantages**

- Minimizes modification of P2T2. P2T2 would be modified only to report the student performance and simulation state information and to receive commands to display or hide visual aids and start, pause, or reset the simulation.
- Eases maintenance problems. Since the modification of P2T2 would be minimized, the problem of controlling the source code of P2T2 is minimized.
- Modularizes the IT. Implementing the IT as a separate process and program will make it easier to move it to another platform or simulation.

### **3.2.2 Disadvantages**

- Limits part-task training. Since the modification of P2T2 would be strictly limited, this would limit the part-task training's ability to modify the simulation.
- Performance penalties. The IT and P2T2 running as separate process would incur processing overhead for two processes instead of one. This might be alleviated if the IT is moved to a different platform communicating via a network.
- Interprocess communication. Interprocess communication might limit the speed of both the IT and P2T2.

### **3.3 Conclusions**

We have decided to implement the IT and P2T2 as separate programs and processes (see Figure 4). This configuration gives us the most flexibility for the future. The IT could be more easily moved to a different platform or simulation. It would minimize the modification of P2T2. The possible interprocess communication bottleneck can be addressed in several ways: moving the IT to a 386 machine running UNIX and communicating via network, using a faster Silicon Graphics computer, or minimizing the amount off data sent to the IT from P2T2.



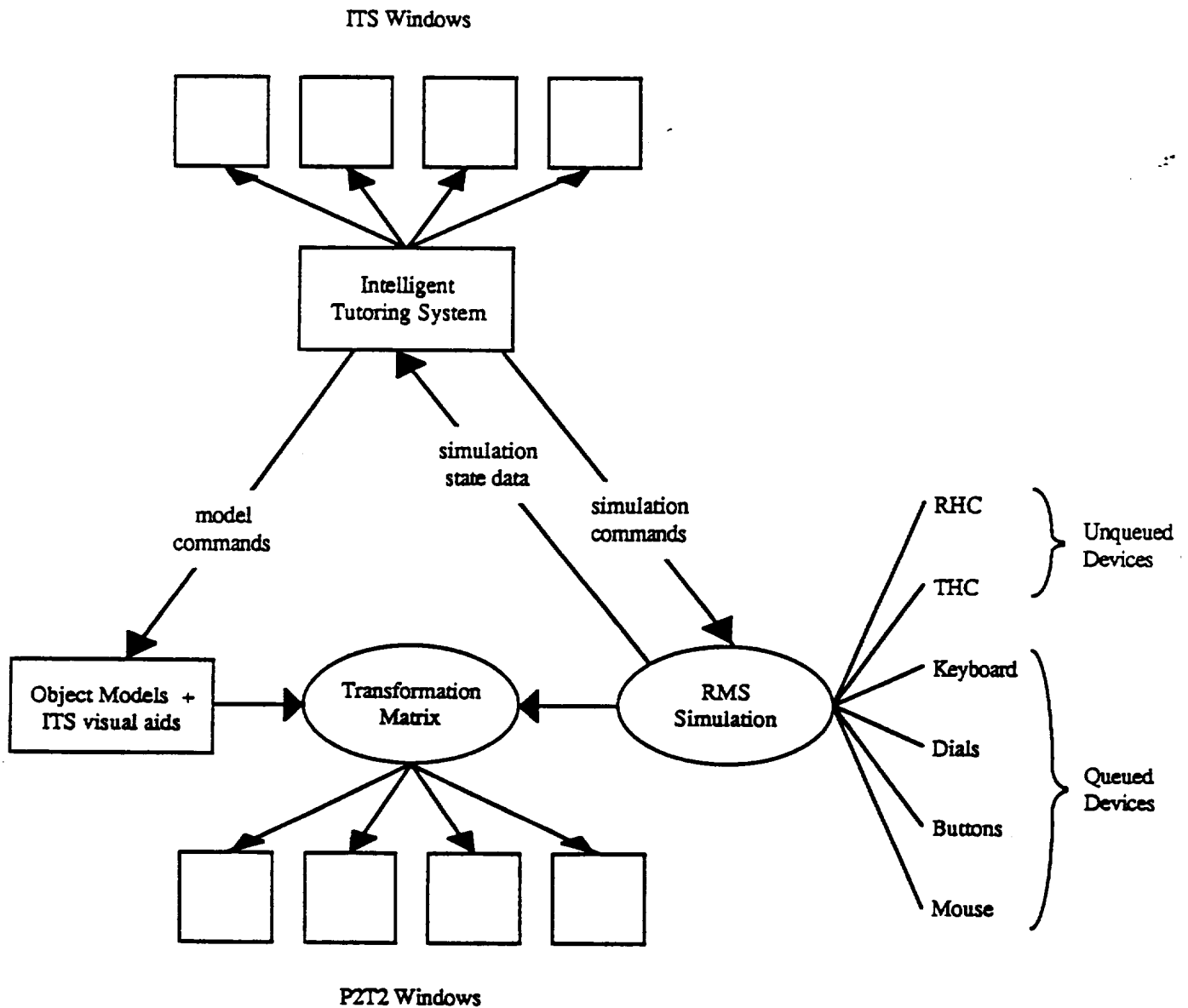


Figure 4. Overview of IT and P2T2 as separate processes

#### 4. Interprocess Communication between P2T2 and The Intelligent trainer

The Intelligent Trainer (IT) and the Prototype Part Task Trainer (P2T2) will communicate by using a message-passing scheme provided by the Silicon Graphics IRIX software. The main goals in communicating between the processes are to intervene as little as possible with P2T2 software, and to make the communication as general as possible, so that the processes can run on different machines without affecting the implementation.

We chose message passing over shared memory for the following reasons. Although shared memory would be more efficient than message passing, it requires that the communicating processes be on the same machine, accessing the same memory. It would also require redefining existing data in P2T2 to be shared, which is an unacceptable intrusion into P2T2 code. In message passing, the concept is general enough that if the IT were moved to a different machine, redefining how the message queue works would not affect the messages being passed. Therefore, the communication will look the same to both P2T2 and the IT regardless of how it is done.

The capabilities required in communicating are the ability to send the IT both data and user actions, and the ability to send P2T2 commands to be run. To do this, we will create files of code that will be added to P2T2 that will send data and actions, and process the commands sent by the IT. Note that these files will be part of P2T2 and not part of the IT, since they will run with the P2T2 process. The IT code that receives data and actions and sends commands will be put in a separate directory, with the IT code. Every effort will be made to limit insertions into actual P2T2 code. Subroutines will be used instead of multiple lines of code, and `#ifdef` compiler statements will be put around those subroutines so that P2T2 can be recompiled without the additions necessary for the IT to run.

Messages from the IT can be sent anytime to P2T2, since we do not have to worry about limiting code insertion on the IT. On P2T2, we will process the message queue once per `Exec()` loop. The `Exec()` loop is P2T2's high-level loop that processes all the user inputs from various devices and translates them into graphics commands. It is here that all messages will be sent to the IT, and all command messages from the IT will be processed.

##### 4.1 Sending and Receiving Data

Data will be handled as follows. We will create copies in P2T2 of all "significant" data structures that the IT needs for processing. There are many data variables in P2T2, but we will be copying only those variables which are of interest to the IT, such as the arm joint angles, the point of resolution position and attitude, etc. When one of these P2T2 variables changes, the change will be made in the copy of the data, and the changed value will be sent to the IT. Since the variables of interest to the IT will change for each part task, only some of the significant data will be marked as active, depending on which part task is being run. Only active data will then be sent. Also, a "noise" factor will be specified by the IT, which will tell P2T2 not to send a piece of data if it has not changed by more than the noise level, which will be different for each piece of data. (Note that the noise level and the active data can be combined by setting inactive data noise level impossibly high. If efficiency becomes a problem, this will be done.)

A more intuitive approach would have been to have the IT send a message to P2T2 each time it needs a data value, and then have P2T2 send the correct value back. However, on the IT side of processing, it would be very inefficient to have to send out a message and wait for a reply every time it wants access to data value, especially a rapidly-changing one such as the position of the arm when it is moving. Using our approach, the IT merely needs to check the message queue, and if a message is there with a new value for a piece of data, then it uses that; otherwise, it uses the old value.

## **4.2 Sending and Receiving Commands**

Commands will be sent to P2T2 from the IT to do such things as:

- load in a new model
- position the arm
- enter the correct modes
- add or remove graphic objects
- enable or disable devices
- control which windows and cameras are visible
- reset the system

Most commands will probably be sent in order to set up a scenario for the student to run. However, they can be sent by the IT at any time and will be processed by P2T2 during the next Exec() cycle. P2T2 code insertion should be minimal for commands, unless the command cannot be processed during an Exec() cycle.

## **4.3 Sending User Actions**

User actions are processed by P2T2 using a queue that tells which device the user acted on and which value was changed on the device. Unfortunately, there is no "peek" function that will allow us to see the value on the queue without removing it from the queue. Removing it from the queue and then adding it back means it will lose its IT place and another action could slip in ahead of it, changing the meaning of the first command (for example, panning left on a camera, then changing cameras).

The code to process the queue is split up among many subroutines in P2T2, so we may have to insert code into all those subroutines to send the actions off to the IT. This is complicated by the fact that we are not yet certain which user actions will be needed by the IT to make judgements about the student's capabilities. Since this part of the communication is still nebulous, we will have to invoke some hand-waving and say that we will send actions in a way similar to the way we send data; that is, we will keep a list of actions that the IT wants to keep track of, and send them when they occur. However, this approach could change when more is known of what the IT needs.

## 5. IMPLEMENTATION GUIDELINES

Given our approach of implementing the Intelligent Tutor as a separate process and the constraint of minimizing the modifications to P2T2, we have set out a few guidelines that we will follow. These guidelines will help when incorporating new versions of P2T2 and ease maintenance of P2T2's source code.

1. Changes to P2T2 source code will be documented as follows:

```

      .
      .
      .
P2T2 code
      .
      .
      .
/* ITS MODIFICATION BEGINS HERE */
      .
      .
      .
IT code inserted here
      .
      .
      .
/* ITS MODIFICATION ENDS HERE */
      .
      .
      .
P2T2 code continues
      .
      .
      .
```

The comments "ITS MODIFICATION BEGINS HERE" and "ITS MODIFICATION ENDS HERE" can be used with the UNIX "grep" command to locate modifications to P2T2's source code.

2. The number of modifications to P2T2 should be minimized.
3. Modifications should be grouped together where possible.
4. The number of lines added to P2T2 source code should be minimized. If they are extensive, they should be turned into a function in a separate source file.

#### **4.5 Results of Performance Analysis: Node Propagation**

Performance and Error Analysis of the testing part task result in a set of nodes in the Student Model that are considered mastered or misused. The states of the nodes are updated. Since the Student Model is a hierarchical model of the student's knowledge, the supporting subskills and subconcepts of each node that has been judged to be mastered or misused are also updated. If a node has been mastered, all its subskills and subconcepts are considered mastered. If a node has been misused, all its subskills and subconcepts are marked "suspect" as possible causes for the error.

After the node state propagation has changed the Student Model, the maintenance loop returns to the test-or-tutor decision. If a misconception has been isolated, it is tutored; otherwise, another test is chosen, probably further down in the domain hierarchy.

#### **4.6 Instructional Assignment**

\*\*\*\*TBD



**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix F: Modifications to the P2T2**

---

<b>APPENDIX F: MODIFICATIONS TO THE P2T2</b>
--





**Appendix: Modifications to P2T2 Source Code as of 12/11/90**

---

The following listing is a list of the P2T2 source files that have been modified.

---

<b>File</b>	<b>Modification starts at line:</b>	<b>Ends at line:</b>
control.c	883	885
control.c	930	975
control.c	987	992
exec.c	518	523
main.c	140	147
main.c	350	353
wins2.c	1597	1631
wins2.c	2280	2288
wins2.c	2555	2648

---



**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix G: Performance Analysis Design**

---

<b>APPENDIX G: PERFORMANCE ANALYSIS DESIGN</b>
--



Section	Section Description	Page
1.	Introduction .....	3
2.	Purpose.....	4
2.1	Overall System Architecture.....	4
3.	Evaluation of Procedural Performance .....	6
3.1	Monitoring the Simulation.....	6
3.2	Evaluating a Procedure with a Procedural Network.....	7
3.3	Provides multiple levels of abstraction for reasoning .....	7
3.4	Provides flexible framework for procedure recognition .....	7
3.5	Can be used for real-time evaluation .....	10
3.6	Simulation state representation .....	10
4.	Description of Procedural Networks.....	11
4.1	Plan start, plan end nodes .....	11
4.2	Goal nodes and subplans .....	11
4.3	Andsplit, andjoin nodes .....	11
4.4	Orsplit, orjoin nodes .....	11
4.5	Node effects.....	11
4.6	Link predicates.....	12
4.7	Procedural ordering links .....	12
5.	Results of Procedural Evaluation .....	14
5.1	Evaluation of Global Skills.....	14
6.	Implementation of the Procedural Network.....	15

**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**  
**NASA P2T2 Intelligent Trainer Design**  
**Performance Analysis Design Document**  
**August 21, 1990**

**List of Figures**

<b>Figure</b>	<b>Description</b>	<b>Page</b>
1.	Overview of P2T2/IT.....	5
2.	An example of a procedural network.....	9
3.	The structure of a procedural network.....	13

## **1. Introduction**

This document describes the Performance Analysis module of the NASA P2T2 Intelligent Trainer (P2T2/IT). The following documents describe the other five modules and aspects of P2T2/IT:

- P2T2 Intelligent Trainer Design Overview
- P2T2/IT Error Analysis Design
- P2T2/IT Student Model Design
- P2T2/IT Deficiencies Analysis Design
- P2T2/IT Instructional Assignment Design
- Modifications to P2T2 Design

Refer to section 5 of the "P2T2 Intelligent Trainer Design Overview" for a discussion of how each module relates to each other.

## 2. Purpose

Performance Analysis conducts an initial analysis of the student's manipulation of the Remote Manipulator System (RMS). Performance Analysis only analyzes the student's current performance and does not look for possible historical causes of the student's behavior. The historical analysis of the student is left to Error Analysis and the Student Model modules.

To do this Performance Analysis must contain use a representation of expert performance to base its analysis of the student. Performance Analysis must analyze the student over two different sets of criteria. Performance Analysis must analyze how well the student performed a procedure or task in the RMS domain and how well they followed safety, efficiency, and accuracy constraints while performing the task or procedure. For example, Performance Analysis must determine how well the student performed a procedure such as grappling a payload in orbit, berthing it in the cargo bay, and stowing the arm by checking if the student performed the correct sequence of actions. Global constraints such as safety, efficiency, and accuracy are monitored over the entire task and cannot be determined just by looking at one step in the procedure or task. Performance Analysis will perform both a procedural analysis and a constraint analysis of the student.

### 2.1 Overall System Architecture

In order to monitor the student, Performance Analysis must examine the simulation state data (student actions, Point of Resolution (POR), movement of the RMS, etc.) and interpret them with respect to the performance of a task and overall safety, efficiency, and accuracy constraints. The results of this interpretation are passed to the Error Analysis module for further analysis. See Figure 1 for an illustration of how Performance Analysis fits into P2T2/IT.



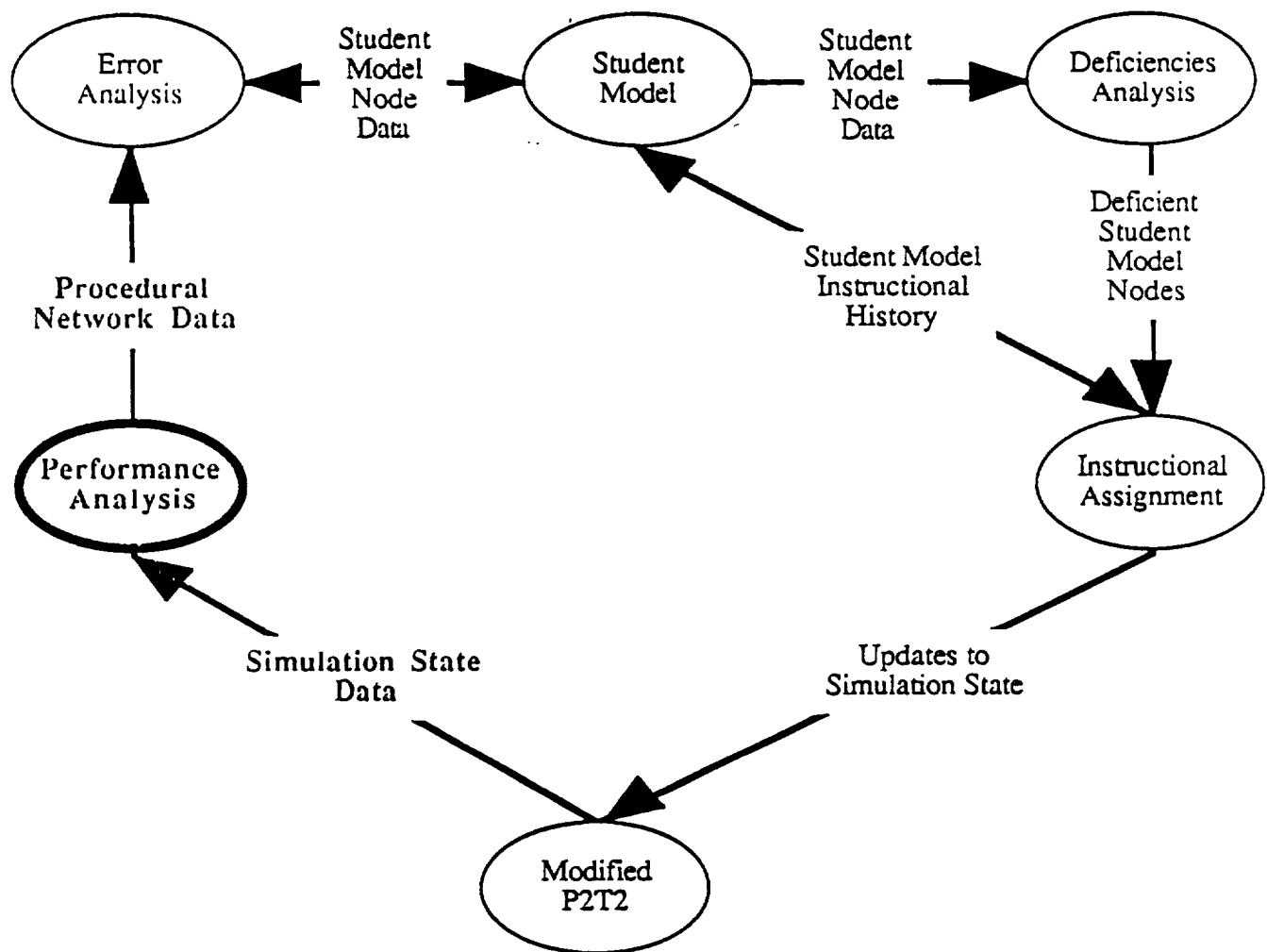


Figure 1. Overview of P2T2/IT

### 3. Evaluation of Procedural Performance

We have chosen to carry out the procedural evaluation with a procedural network<sup>1</sup>. A procedural network is a powerful representation for interpreting a student's actions as he or she is performing the procedure. Briefly, the advantages of a procedural network are:

- can be used for real-time evaluation of the procedure
- multiple levels of abstraction in the procedure allowing the procedure to be evaluated at different levels of detail
- mechanisms for representing the partial ordering of procedures
- a representation of the "world" as it relates to the procedure

Procedural networks have a venerable history in intelligent tutoring systems. Many researchers have chosen procedural networks as a representation of the knowledge domain.<sup>2,3,4</sup> We found that the procedural network is a convenient and powerful representation in Phase I of the project.<sup>5</sup>

#### 3.1 Monitoring the Simulation

The Performance Analysis module monitors the student's performance and compares it to the procedural network which represents the "correct" way of performing the procedure. The procedural evaluation only analyzes the student's current performance and does not look for possible historical causes of the student's behavior. The historical analysis of the student is left to Error Analysis and Deficiencies Analysis modules. The procedural network detects any mistakes the student makes during the procedure and reports them to Error Analysis where they will be examined further.

The Performance Analysis module also maintains the procedural network. Since we are using the procedural network as the representation of the expert's domain, the procedural network must monitor the state of the simulation as the student changes it.

#### 3.2 Evaluating a Procedure with a Procedural Network

The representation of the procedure by the procedural network is critical to the success of the diagnostic modules Performance Analysis and Error Analysis. The representation must provide a framework which allows reasoning about the relationship among the

---

<sup>1</sup>Sacerdoti, D. (1977). *A Structure for Plans and Behavior*. Elsevier North-Holland, New York.

<sup>2</sup>VanLehn, K.; and Brown J. S. (1980). Planning Nets: a representation for formalizing analogies and semantic models of procedural skills. In Snow, R.; Frederico, P.; and Montague, W. (Eds.) *Aptitude, Learning and Instruction: Cognitive Process Analyses*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

<sup>3</sup>Rickel, J. (1988). An Intelligent Tutoring Framework for Task-Oriented Domains. *Intelligent Tutoring Systems June 1 - 3, 1988*, Montreal, pp.109-115.

<sup>4</sup>Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*, Morgan Kaufman Publishers, Los Altos, California.

<sup>5</sup>NASA Phase I SBIR final report on the Intelligent Procedure Evaluator

various parts of the procedure, the sequencing of the various tasks in the procedure, and how the tasks affect the "world" (the state of the simulation). Moreover, since the procedural network is a data structure and is not encoded in rules, the procedural network is also accessible to reasoning by other modules. As data, it can easily be modified and adapted to changing procedures.

Procedural networks can be constructed dynamically but P2T2/IT will not dynamically construct its procedural network in our initial prototype. There are two reasons for this. Since we have chosen to restrict knowledge acquisition to a small set of tasks in the RMS domain it is not necessary to build the procedural networks dynamically. Second, the dynamic construction of procedural networks is computationally difficult involving techniques such as plan criticism and plan optimization. Dynamically constructed procedural networks might contain flaws that would affect their use during diagnosis. Our architecture does not preclude the dynamic construction of procedural networks if they are needed in the future.

### 3.3 Provides multiple levels of abstraction for reasoning

The hierarchical nature of procedural networks makes them ideal for reasoning about the procedure at different levels. Student diagnosis can measure skills and performance at different levels of the procedure. For example, we might want to measure an overall quantity like the time to perform a section of the procedure. The hierarchical nature of the procedural network allows us to measure it at different levels without examining and interpreting the individual actions that accomplish that section of the procedure.

### 3.4 Provides flexible framework for procedure recognition

One of the most difficult tasks in procedure evaluation is understanding the student's progress through the procedure. Often, procedures contain some flexibility in the order of steps. Procedures often offer opportunities for the student to correct his mistakes and continue with the procedure. A rigid representation of the procedure will result in a rigid analysis of the student. If the plan contains tasks that can be performed in any order, we can't rely on a step-by-step ordering of the student's actions for evaluation. The procedural network's "orsplit" nodes are a good representation for such flexible plans (section 4.4 discusses orsplit nodes). The procedural network's andsplit nodes can represent the strict ordering of plan steps (section 4.3 discusses andsplit nodes).

As an example, suppose a section of the procedural network contains this plan of independent tasks:

- Reset the widget A (press button 1)
- Turn on widget B (turn knob 1 to "on")
- Prepare widget C (accomplished by subplan)
  - Set gizmo 1 (turn knob 2 to "5")
  - Turn off gizmo 2 (turn switch 1 to "off")

Suppose that the widgets and gizmos are independent mechanisms: manipulating one widget does not affect the operation of any of the others. If the plan were executed in strict sequence it would result in the following sequence of actions:

press button 1  
turn knob 1 to on  
turn knob 2 to 5  
set switch 1 to off

But suppose the student executes the actions in this order:

set switch 1 to off  
press button 1  
turn knob 2 to 5  
turn knob 1 to on

The procedural network can interpret this sequence of actions as accomplishing the plan even the actions are not in the strict ordering. See Figure 2 for an illustration of this plan.

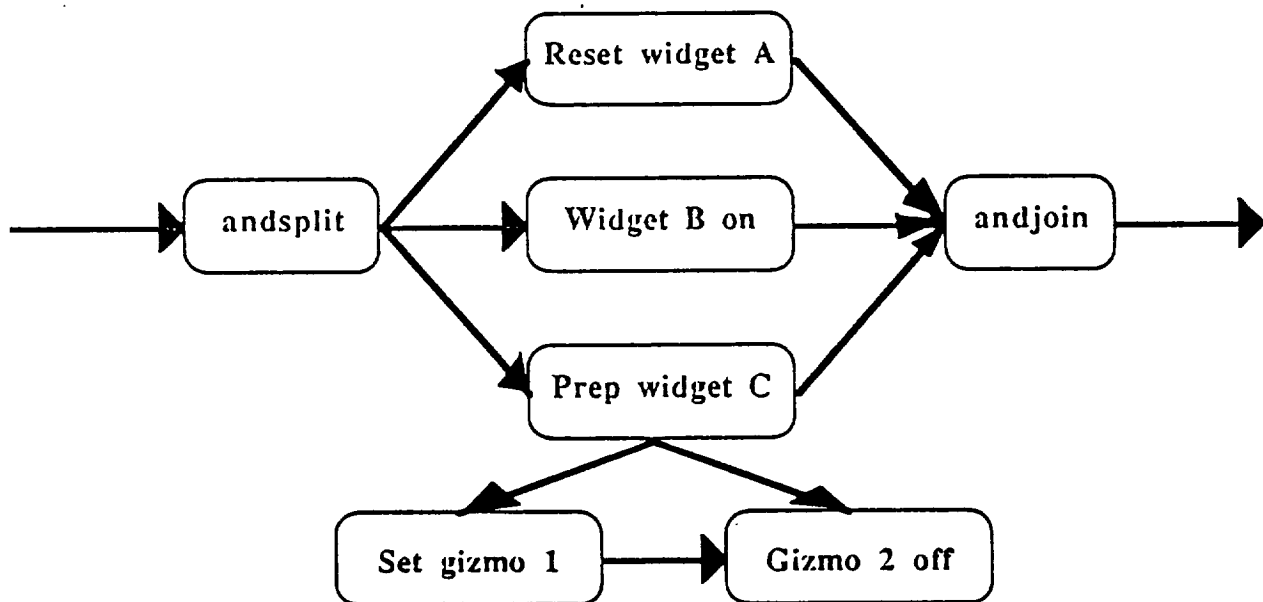


Figure 2. An example of a procedural network

### 3.5 Can be used for real-time evaluation

Another advantage in using procedural networks as a representation is that they can be used to evaluate the procedure as it is performed. Evaluating the student becomes a question of parsing the student's actions, monitoring the state of the simulation, and comparing them to the procedural network. This can be done in a top-down fashion.<sup>6</sup> This will be a great benefit in the real-time evaluation of the student's performance.

It is important to note that the state of the procedural network at any point in time is a complete description of the state of the simulation as well as the state of procedure. As the student moves through the procedure his actions are interpreted by the procedural network by how he or she has changed the state of the simulation.

### 3.6 Simulation state representation

As mentioned above, the procedural network is not only a representation that describes the procedure but also the state of the simulation. The procedural network describes how each step of the procedure affects the state of the world. This description of the procedure allows reasoning by the modules of P2T2/IT on the effects and relationship of parts of the procedural network and how they affect the interpretation of the student's performance.

---

<sup>6</sup>Rickel, J. *ibid.*

#### 4. Description of Procedural Networks

Procedural networks were first characterized by Sacerdoti.<sup>7</sup> They are closely related to augmented transition networks. The procedural network is composed of four basic classes of nodes described below.

The procedural network is ordered by its links among nodes. Nodes may have predecessors, successors, a parent and children (see Figure 3). The successor and predecessor links order the procedure. Parent and child links denote subprocedures that must be executed to achieve the effects of the parent procedure. The parent and child link allow the procedural network to be ordered hierarchically.

##### 4.1 Plan start, plan end nodes

Plan start and plan end nodes are delimiters of the plan. They are used for both plans and subplans.

##### 4.2 Goal nodes and subplans

Goal nodes organize the procedural networks hierarchically. Goal nodes are accomplished by subplans that are linked as children. In the example illustrated in Figure 2 the node "Prep widget C" is a goal node that is accomplished by the subplan "Set gizmo 1" and "Gizmo 2 off" (note: the plan start and plan end nodes have been eliminated from the illustration for clarity).

##### 4.3 Andsplit, andjoin nodes

Andsplit and andjoin nodes delimit a collection of steps which may be evaluated independently. The procedural network in Figure 2 illustrates an andsplit and andjoin in a procedure. The steps in Figure 2 "Reset widget A", "Widget B on", and "Prep Widget C" are independent steps that may be performed in any order. The andsplit and andjoin nodes themselves delimit the independent steps.

##### 4.4 Orsplit, orjoin nodes

Orsplit and orjoin nodes are similar to the andsplit/andjoin nodes. They delimit a set of steps only one of which must be performed successfully. The orsplit and orjoin delimit the steps.

##### 4.5 Node effects

Each node in the procedural network contains lists of effects on the world. They represent the changes to the simulation state model caused by completing the procedure step. The effects are used to recognize and interpret student actions as they change the state of the simulation.

---

<sup>7</sup>Sacerdoti, D. *ibid.*

#### **4.6 Link predicates**

Since the procedural network is not constructed dynamically, link predicates are used to control branches of the procedural network based on the state of the simulation. For example, a branch for an error correction procedure may be enabled or disabled depending on the state of the equipment the link predicate is monitoring.

#### **4.7 Procedural ordering links**

Procedural ordering links are used to represent ordering information not captured by the successor and predecessor links. In our previous example illustrated in Figure 2, the steps "Reset widget A", "Widget B on", and "Prep widget C" can be performed in any order but we want to require the student to execute them in the order "Reset widget A", "Widget B on", "Prepare widget C".

The procedural ordering links are used to express ordering of the procedure not required by the machine states. The procedural ordering is kept separate from the machine state representation.



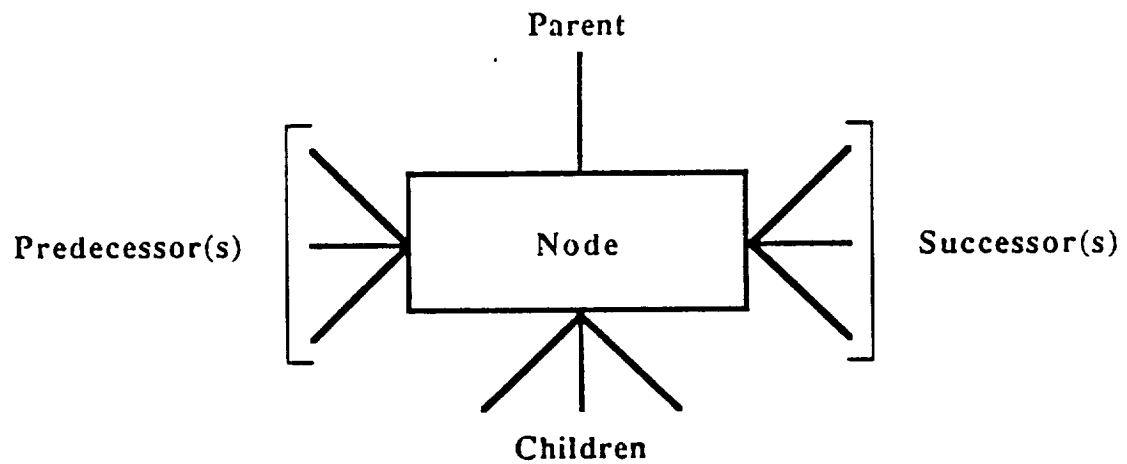


Figure 3. The structure of a procedural network

## **5. Results of Procedural Evaluation**

As the student works through the procedure, Performance Analysis will monitor his or her progress. If the student makes a mistake and does not complete a procedure step then Performance Analysis will detect this as an unsatisfied node in the procedural network. Other mistakes might be detected as the student attempts to work through inactive branches of the procedural network. Detecting an error in the procedural network does not provide an explanation in terms of the Student Model. The mistake might be due to an oversight or ignorance of how to correctly perform the procedure.

Performance Analysis does not attempt to explain the mistakes it observes in the procedural network. Error Analysis will classify these mistakes in terms of the state of the procedural network and pass them to the Student Model and Deficiencies Analysis for further diagnosis and explanation. The Error Analysis process is described in "P2T2/IT Error Analysis Design". The Student Model is described in "P2T2/IT Student Model Design". The Deficiencies Analysis is described in "P2T2/IT Deficiencies Analysis Design".

### **5.1 Evaluation of Global Skills**

As we mentioned before, not every skill in the RMS domain can be measured by evaluation with a procedural network. Constraints like safety, efficiency, and accuracy are used in every procedure and must be observed throughout the entire procedure. They cannot be determined by evaluating one step or section of a procedure in isolation. The constraints of accuracy, efficiency, and safety are global evaluations of the student and are best measured over the entire performance of the student.

Thus global skills must be measured independently of the procedural evaluation done by the procedural network. A skill like "safety" could be measured by counting the number of times the student violates the safety guideline "vernier movement should be used when the RMS is within five feet of the payload or orbiter". Appropriate heuristics will be developed to measure the skills of safety, efficiency, and accuracy.

**6. Implementation of the Procedural Network**

Procedural networks lend themselves to object-oriented implementations. We have obtained a C++ compiler for the Silicon Graphics computer. The compiler is based on cfront 1.2. Both P2T2 and the CLIPS inference engine are written in C. C++ is compatible with C so we will not have any difficulty integrating P2T2 (written in C), CLIPS (written in C) with the IT (using C++).



**GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.**

**NASA SBIR Phase II Final Report**

**Contract #NAS 9-18170 "Enhanced Intelligent Evaluation System for Simulation"**

**August 15, 1991**

**Appendix H: Error Analysis Design**

---

<b>APPENDIX H: ERROR ANALYSIS DESIGN</b>
--



Section	Section Description	Page
1.	Introduction .....	3
2.	Purpose.....	4
2.1	Overall System Architecture.....	4
3.	Performance Analysis .....	6
3.1	Procedural Evaluation .....	6
3.2	Global Skills and Constraints Evaluation.....	6
3.3	Results of Performance Analysis .....	7
4.	Error Analysis: Classifying Unsatisfied Procedures.....	8
4.1	Invalid action.....	8
4.2	Problem violation .....	8
4.3	Irrelevant plan .....	8
4.4	Incorrect plan.....	9
4.5	Ordering violation .....	9
5.	Error Analysis Interpreting the Error Classifications .....	13

GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.  
NASA P2T2 Intelligent Trainer Design  
Error Analysis Design Document  
August 21, 1990

List of Figures

Figure	Description	Page
1.	Overview of P2T2/IT.....	5
2.	Irrelevant Plan Example.....	11
3.	Ordering Violation Example.....	12



## **1. Introduction**

This document describes the Error Analysis module of the NASA P2T2 Intelligent Trainer (P2T2/IT). The following documents describe other modules and aspects of P2T2/IT:

- P2T2 Intelligent Trainer Design Overview
- P2T2/IT Performance Analysis Design
- P2T2/IT Student Model Design
- P2T2/IT Deficiencies Analysis Design
- P2T2/IT Instructional Assignment Design
- Modifications to P2T2 Design

Section 2 of this document gives a general overview of how Error Analysis fits into the overall architecture of P2T2/IT. Section 3 briefly describes how Performance Analysis is carried out and how Error Analysis builds on the results of Performance Analysis. Section 4 describes how Error Analysis interprets the student errors located by performance. Section 5 shows Error Analysis determines how the historical model of the student, maintained by the Student Model, should be modified.

## 2. Purpose

Error Analysis interprets the evaluation of the student's current performance done by Performance Analysis and determines how the historical record of the student maintained by the Student Model must be changed. Error Analysis seeks to explain in terms of the student's knowledge and skills the errors observed by Performance Analysis.

### 2.1 Overall System Architecture

Figure 1 illustrates Error Analysis in the overall architecture of P2T2/IT. Error Analysis receives the procedural network monitored by Performance Analysis. Error Analysis then seeks to explain the errors observed by Performance Analysis in terms of the student's skills and concepts maintained by the Student Model. Finally, Error Analysis gives the Student Model a list of skills and concepts it has determined the student has misused or mastered. The Student Model uses this list to maintain its historical model of the student.

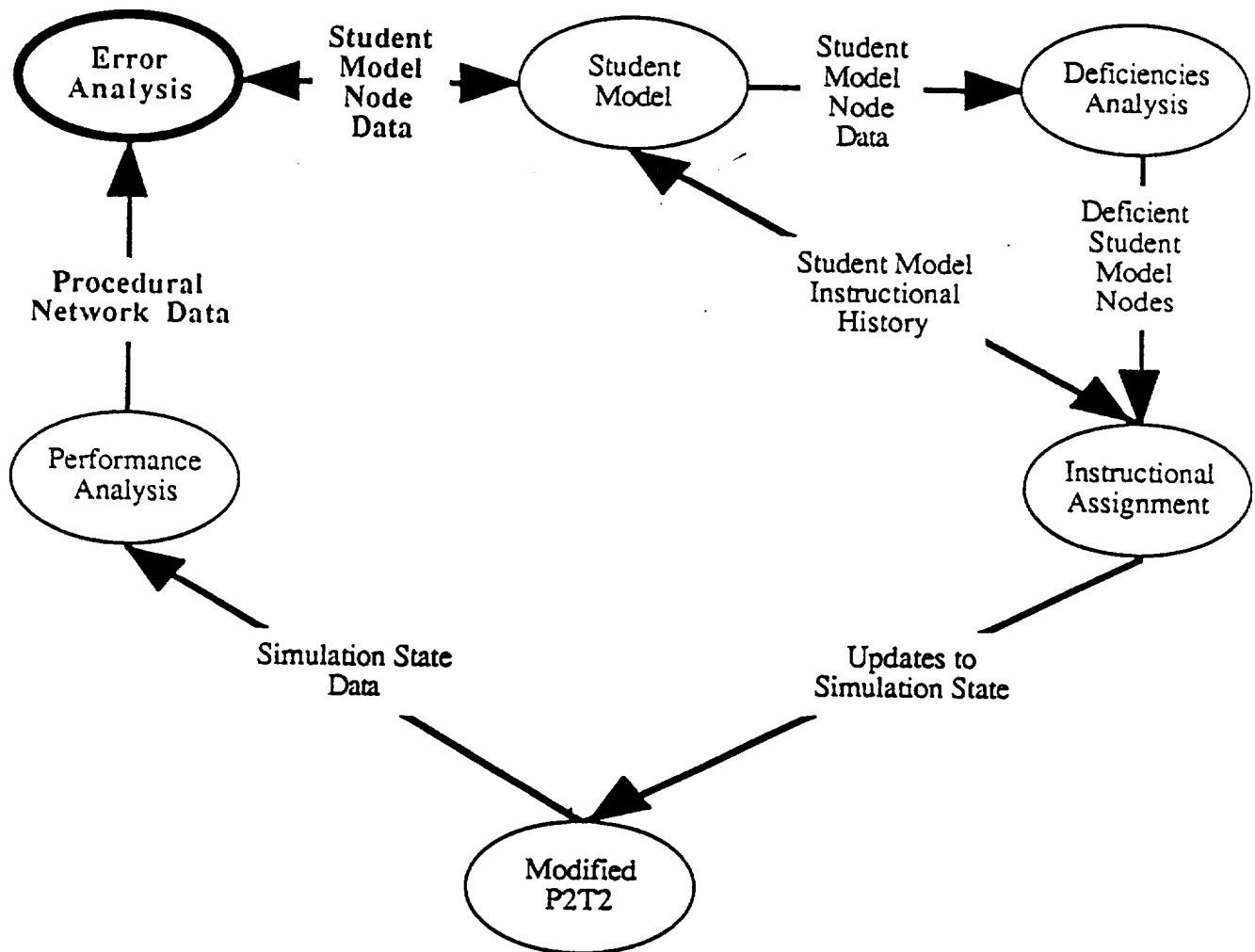


Figure 1. Overview of P2T2/IT

### 3. Performance Analysis

Performance Analysis performs an initial analysis of the student's manipulation of the RMS. Performance Analysis only analyzes the student's current performance and does not look for possible historical causes of the student's behavior.

Performance Analysis must analyze the student over two different sets of criteria:

- procedural correctness
- global skills and constraints correctness

Performance Analysis must determine how well the student performed a procedure or task in the RMS domain and how well they followed safety, efficiency, and accuracy constraints while performing the task or procedure. For example, Performance Analysis must determine how well the student performed a procedure such as grappling a payload in orbit, berthing it in the cargo bay, and stowing the arm by checking if the student performed the correct sequence of actions. Global constraints such as safety, efficiency, and accuracy are monitored over the entire task and cannot be determined just by looking at one step in the procedure or task. Performance Analysis will perform both a procedural analysis and a constraint analysis of the student.

#### 3.1 Procedural Evaluation

To perform an evaluation of the student's performance of a procedure or task, Performance Analysis uses a "procedural network"<sup>1</sup>. The procedural network is a representation of the steps of the procedure, the order in which they are performed, and how they affect the state of the simulation (in this case P2T2).

The procedural network is a close relative of the augmented transition network. It imposes a partial order on the procedure with "andsplit" and "orsplit" nodes, link predicates, and procedural ordering links. The procedural network is described in detail in the "P2T2/IT Performance Analysis Design".

#### 3.2 Global Skills and Constraints Evaluation

While the procedural network can evaluate and interpret the student's actions as they perform a procedure or task, the procedural network monitors is geared towards a step-by-step analysis. Global constraints such as safety, efficiency, and accuracy are monitored over the entire task and cannot be determined just by looking at one step in the procedure or task. Therefore, global skills and constraints are monitored separately

#### 3.3 Results of Performance Analysis

As the student works through the procedure, Performance Analysis will monitor his progress with the procedural network. If the student makes a mistake and does not complete a plan step, Performance Analysis will detect this as an unsatisfied node in the procedural network. Other mistakes might be detected as the student attempts to work

---

<sup>1</sup>Sacerdoti, D. (1977). *A Structure for Plans and Behavior*. Elsevier North-Holland, New York.

through inactive branches of the procedural network. Error Analysis will then classify these mistakes in terms of the state of the procedural network and attempt to explain them in light of the historical model of the student.

#### 4. Error Analysis: Classifying Unsatisfied Procedures

As mentioned above, student errors can be classified in terms of the state of the procedural network. Performance Analysis can determine when a student does not perform a step. The classification of unsatisfied procedures is due to Rickel<sup>2</sup> but the "ordering violation" is due to our addition of the procedural ordering links. This classification is an initial explanation of the cause of student error.

##### 4.1 Invalid action

This is an action that the student has taken that is not valid anywhere in the procedural net. Since the procedural network characterizes all possible paths through the procedure and all the possible actions that might be taken somewhere in the procedure, Error Analysis can detect any action that does not fall on a path.

##### 4.2 Problem violation

A student may take actions that are appropriate for a goal but are inappropriate for the initial state of the world.

For example, suppose we have a procedure:

1. Power up the widget (goal)
  - (if widget is type A)
    - 1.1 Set power switch to "on"
    - 1.2 Press widget reset button
  - (if widget is type B)
    - 1.3 Set widget dial to "0"
    - 1.4 Set power switch to "start"
    - 1.5 Press widget reset button
    - 1.6 Set power switch to "on"

Suppose the student was told the widget is type A. If he performs any of the steps 1.3 - 1.5 he was made a "problem violation".

##### 4.3 Irrelevant plan

Since we are not dynamically constructing the procedural network, we will use link predicates to disable parts of the procedural network that will not be needed. Error Analysis can detect if the student attempts to execute these disabled branches and report them as "irrelevant plans".

---

<sup>2</sup>Rickel, J. An Intelligent Tutoring Framework for Task-Oriented Domains. *Intelligent Tutoring Systems June 1 - 3, 1988*, Montreal, pp.109-115.

#### 4. Error Analysis: Classifying Unsatisfied Procedures

---

For example, suppose we have the following procedure (see Figure 2 for an illustration).

1. Prepare gizmo (goal)
  - (if gizmo status is "error")
    - 1.1 Set gizmo power button to "off"
    - 1.2 Set gizmo power button to "on"
    - 1.3 Press gizmo reset button
  - (if gizmo status is "ok")
    - 1.4 Press gizmo button 1
    - 1.5 Press gizmo button 2
    - 1.6 Set gizmo switch to "on"

The "if" statements represent link predicates that enable or disable branches in the procedural network. If the student attempts any of the steps in the error subprocedure Error Analysis will recognize them as "irrelevant plans".

##### 4.4 Incorrect plan

If the student omits a step in a procedure, Error Analysis can detect this as an unsatisfied node in the procedural network. Error Analysis will classify the missed step as an "incorrect plan".

##### 4.5 Ordering violation

We have added the procedural ordering links to the procedural network to represent ordering of the plan not required by the simulation state. Error Analysis will use these procedural ordering links to detect violations in the ordering of student actions that are not mandated by node effects.

For example, suppose we have the following plan (see Figure 3 for an illustration).

1. Prepare the widget (goal)
  - 1.1 Set switch to "A"
  - 1.2 Press button 1
  - 1.3 Turn dial to "5"

Suppose the switch, button, and dial are independent of each other; the operation of one does not affect the state of the others. This would be represented in the procedural network as an andsplit/andjoin branch. We can add procedural ordering links to represent the fact that we want the steps 1.1, 1.2, and 1.3 performed in strict order. Suppose the student performed the actions in this order:

1. Pressed button 1
2. Turned dial to "5"
3. Set switch to "A"

Error Analysis can diagnose this as a ordering violation error. It will not classify it as an incorrect plan error since the student has not violated the andsplit/andjoin construct in the procedural network.

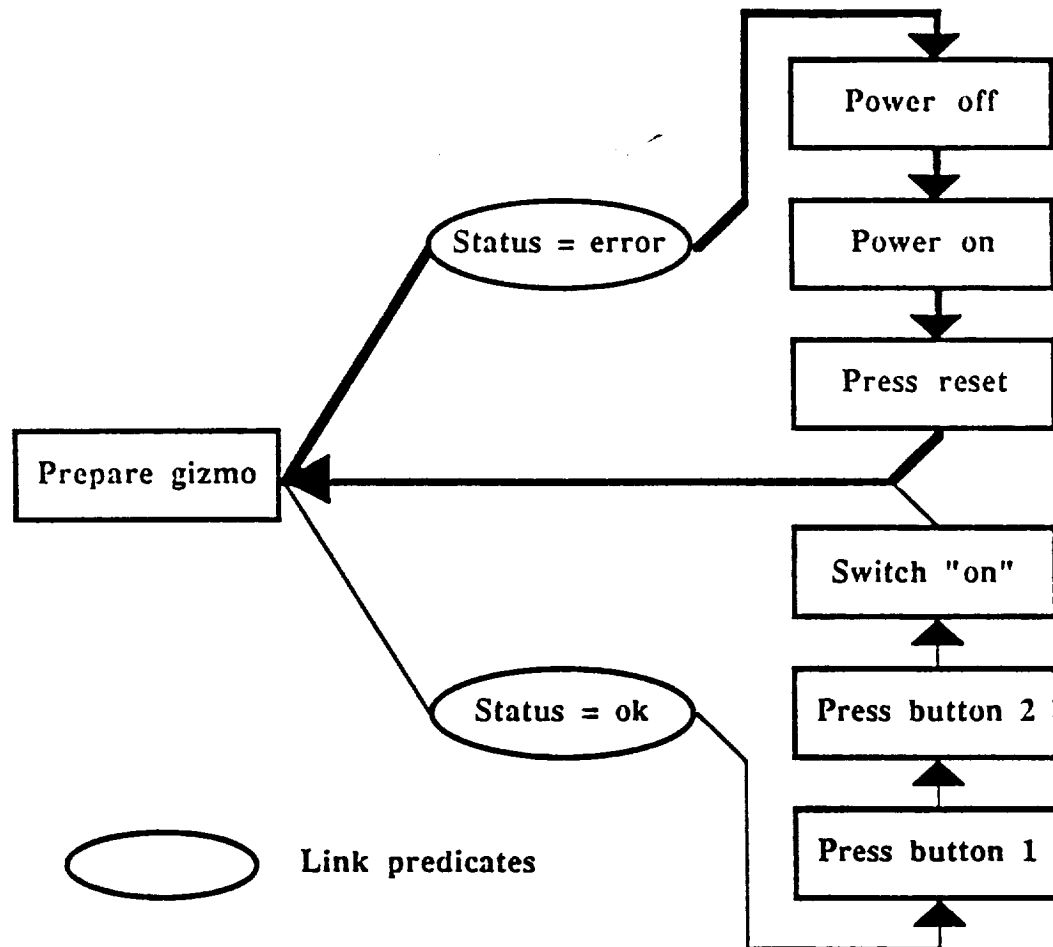


Figure 2. Irrelevant Plan Example



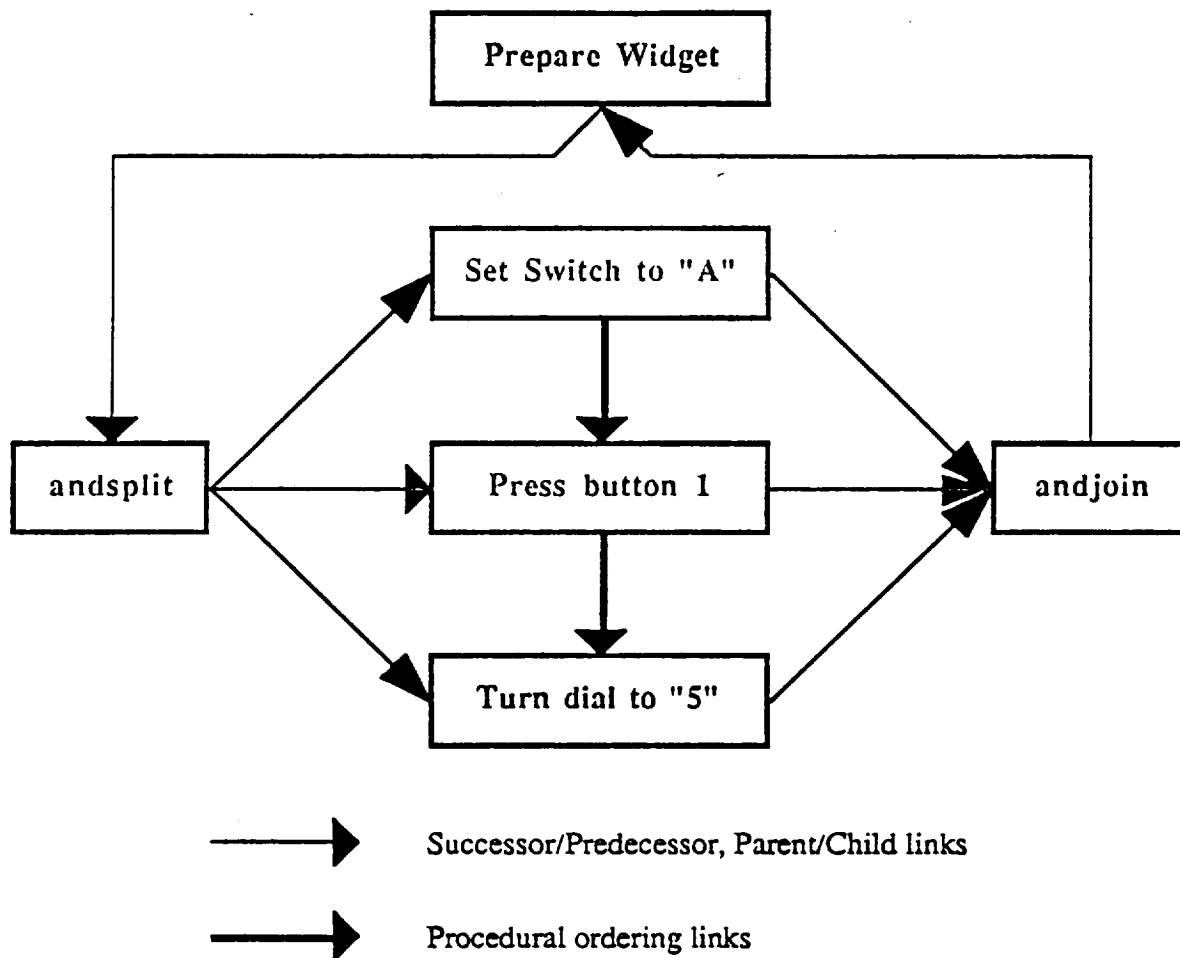


Figure 3. Ordering Violation Example

#### 5. Error Analysis: Interpreting the Error Classifications

The Student Model must maintain the domain hierarchy with the information it receives from Performance Analysis. In the case of the procedural knowledge hierarchy, the Student Model can directly map the results of the procedural network into the procedural knowledge hierarchy since the procedural knowledge hierarchy is a direct reflection of the procedural net. In the case of the skills hierarchy, mapping the results of Compare Solutions will not be so straightforward. Since the elements of the skills hierarchy are defined as skills used throughout the procedure, the Student Model must look at the overall performance of the skill. There probably is not a good way to do this in general.<sup>3</sup> We believe that individual diagnostic techniques can be developed for the skills represented in the skills domain hierarchy. These techniques could be attached to the nodes in the domain hierarchy.

---

<sup>3</sup>Frederiksen, J. and White, B. (1989). *An Approach to Training Based Upon Principled Task Decomposition*. Acta Psychologica 71, North Holland, pp.89-146

<b>APPENDIX I: STUDENT MODEL DESIGN</b>
---



Section	Section Description	Page
1.	INTRODUCTION .....	4
1.1	Basic Assumptions.....	4
1.1.1	Existence of a Domain Hierarchy .....	4
1.1.2	Functions of the Student Model Module.....	6
1.1.3	Relationships with Other P2T2/ITS Modules .....	6
1.2	Sub-Components of the Student Model Module.....	7
2.	REPRESENTING THE DOMAIN HIERARCHY .....	9
2.1	The Domain Hierarchy is Represented as a Network of Nodes.....	9
2.2	Representing the Domain Hierarchy Network.....	10
2.3	Student Model is a Virtual Copy of the Domain Hierarchy.....	10
2.4	Additional Attributes of the Student Model.....	13
3.	STUDENT MODEL INITIALIZATION.....	17
3.1	New Student Model Creation .....	17
3.1.1	Determine Whether Student Is New or Continuing .....	19
3.1.2	Handling New Students.....	19
3.2	Pre-test Student and Update Student Model .....	21
4.	STUDENT MODEL NODE STATE PROPAGATION.....	22
4.1	Downward Propagation of Student Model Node States.....	24
4.2	Upward Propagation of Student Model Node States.....	29
4.3	Evaluating Node State Relationships to Determine Actual Student Errors.....	31
4.4	Examples of Full Node State Propagation .....	31
4.4.1	Node State Propagation Through a Branch.....	32

Figure	Figure Description	Page
1.	Re-orienting the Student Model.....	5
2.	Relationship between the Student Model and other P2T2/ITS Modules.....	6
3.	The Student Model module is a collection of knowledge structures and methods for maintaining the internal states of the knowledge structures.....	8
4.	Network of groups, concepts, and knowledge are linked to the results of Performance and Error Analysis.....	9
5.	The Student Model is a copy of the domain hierarchy.....	12
6.	New student module initialization process.....	18
7.	Building a student model from the domain hierarchy .....	20
8.	Standard symbology used in representing node state propagation .....	23
9.	Downward node propagation at a terminal node.....	26
10.	Downward node propagation through a linear chain of nodes.....	26
11.	Downward node propagation through multiple parallel paths.....	27
12.	Downward node propagation through multiple intersecting paths .....	28
13.	(Part 1 of 4) Example branching propagation.....	32
14.	(Part 2 of 4) Example branching propagation.....	33
15.	(Part 3 of 4) Example branching propagation.....	34
16.	(Part 4 of 4) Example branching propagation.....	35

Table	Table Description	Page
1.	Allowable Student Model node states .....	23
2.	Student model node state down-stream propagation matrix .....	24
3.	Student model node state upstream propagation matrix .....	30

## 1. INTRODUCTION

This document describes the Student Model module of the NASA P2T2/Intelligent Trainer. For an introduction to the functions of the other modules, refer to the following documents describing other modules and aspects of P2T2/ITS:

- P2T2 Intelligent Trainer Design Overview
- P2T2 Intelligent Trainer Performance Analysis Design
- P2T2 Intelligent Trainer Error Analysis Design
- P2T2 Intelligent Trainer Deficiencies Analysis Design
- P2T2 Intelligent Instructional Assignment Design
- Modifications to P2T2 Design

### 1.1 Basic Assumptions

The design of the Student Model Module is based on a set of assumptions about other modules in P2T2/ITS. These assumptions are described in sections 1.1.1 through 1.1.3.

#### 1.1.1 Existence of a Domain Hierarchy

The first assumption is that there is an entity called the "domain hierarchy", which is the network of nodes representing the groups, concepts, sub-concepts, knowledge elements, and skill elements needed by the student to function successfully in the RMS domain. Each node of the domain hierarchy will contain information about the student's mastery or misuse of that skill or concept. The information stored in the domain hierarchy node is presented in Section 2 of this report.

In essence, the domain hierarchy is a knowledge structure which represents a class from which instantiations are made for each student using the system. Therefore, the student model includes both the class (the domain hierarchy) and the particular instantiation. We shall refer to the instantiation of the student model for a particular student as the Student Model in this document.

In representing the domain hierarchy in the Prototype Design report, we depicted the network of nodes as a tree structure with the higher-level nodes shown at the top and the lower level ones at the bottom. For greater clarity in this document, we shall, at times, rotate the tree onto its side and show lower-level nodes on the left and higher-level nodes on the right. This change is shown in Figure 1 on the following page.



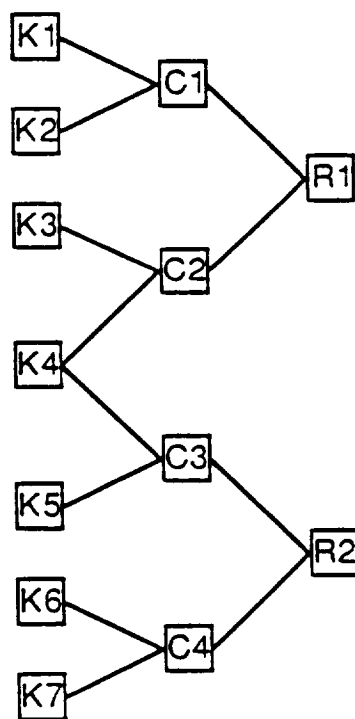
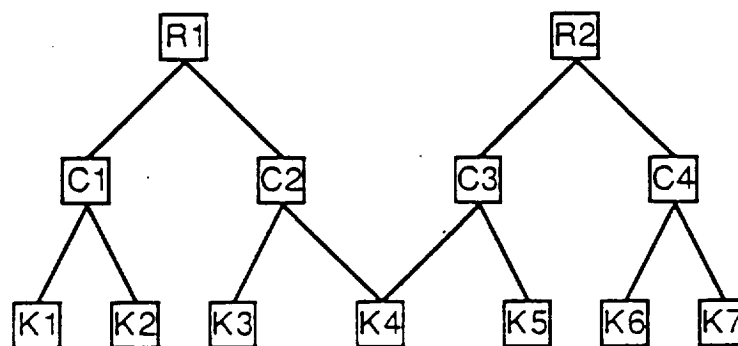


Figure 1. Re-orienting the Student Model

### 1.1.2 Functions of the Student Model Module

The role of the Student Model module is to represent the relationships between the concepts, knowledge elements and skills of the domain, as well as the state of the student's mastery over the domain (e.g., RMS training) and a history of some aspects of the training for this particular student.

### 1.1.3 Relationships with Other P2T2/ITS Modules

The Student Model module receives essential information from the Error Analysis module (via the Performance Analysis module) from which it can update its internal states. This information consists of a collection of facts relating to whether the student correctly used or misused various nodes of the Student Model while performing an assigned part task or whole task. Performance Analysis just analyzes the student's current performance. It does not attempt to explain the student's mistakes or his or her progress over time. Once Performance Analysis has determined what mistakes the student made while performing a task, Error Analysis must determine why the student made mistakes and pass those explanations to the Student Model.

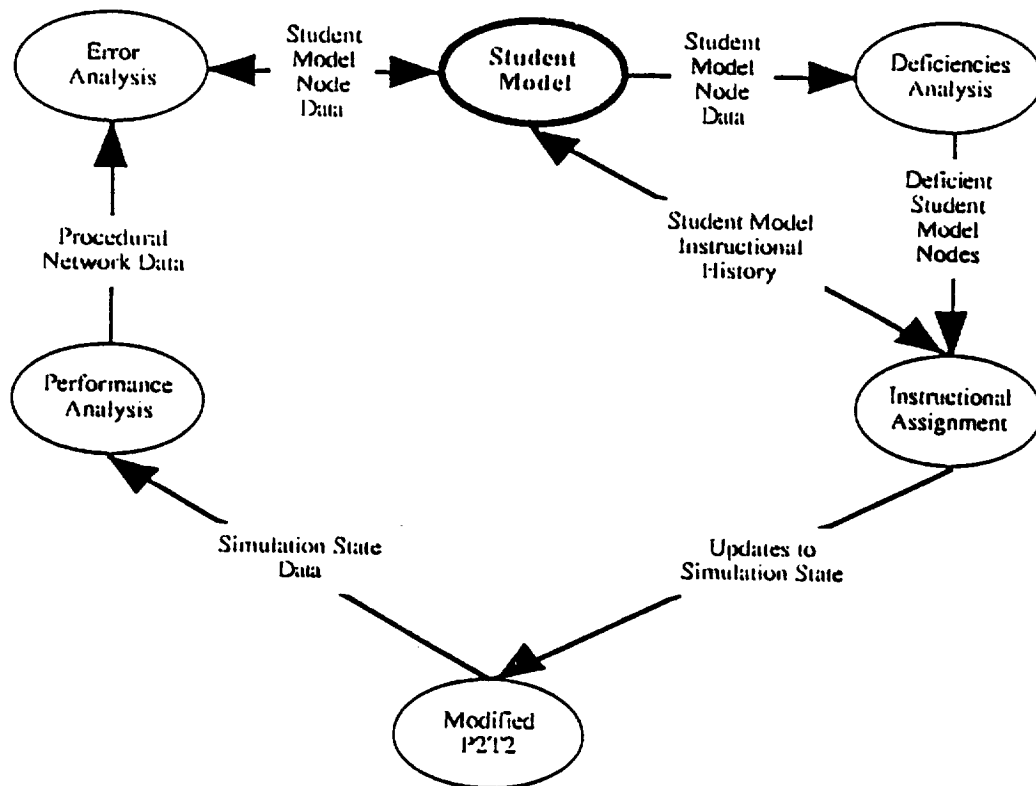


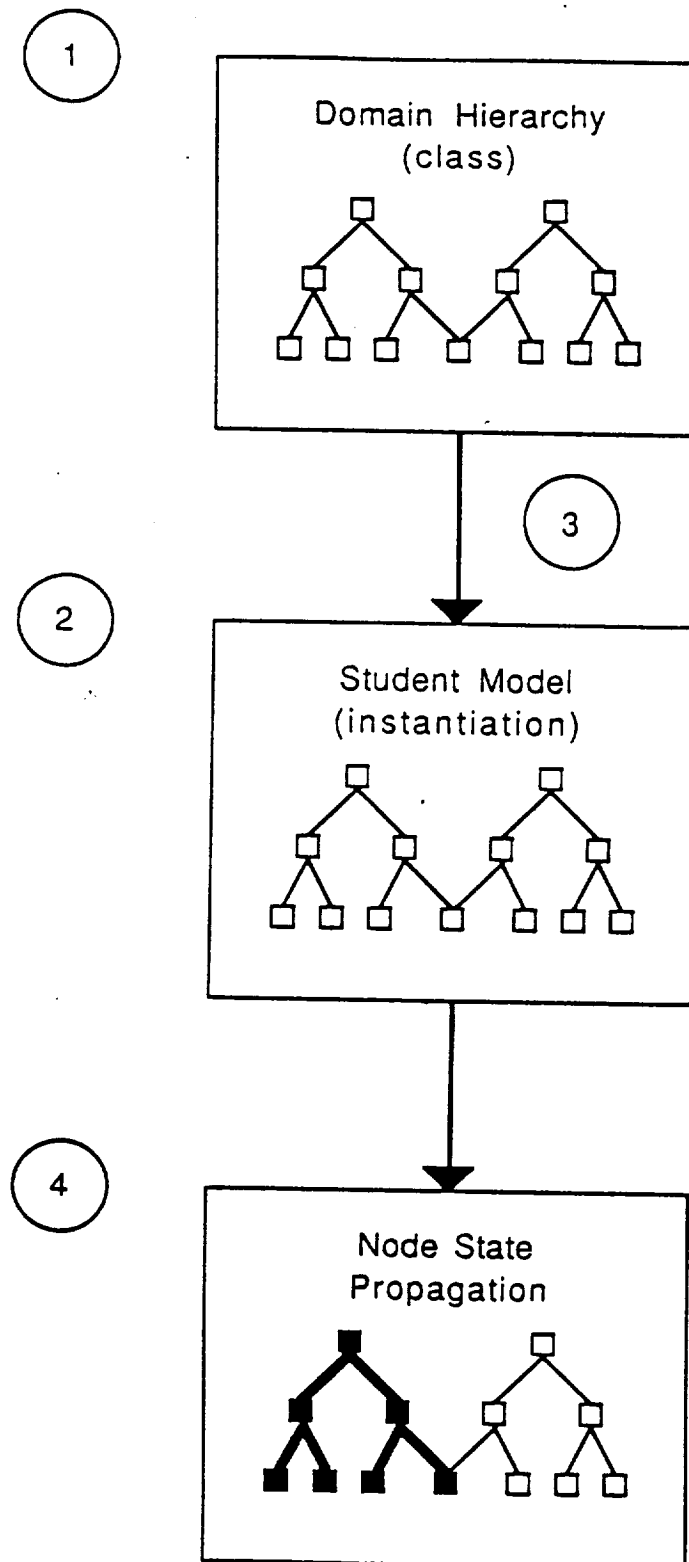
Figure 2. Relationship between the Student Model and other P2T2/IT Modules

### **1.2 Sub-Components of the Student Model Module**

The Student Model module is both a knowledge structure (the hierarchy of concepts, knowledge, and skill elements of the domain) and methods for maintaining (initializing and updating) the internal states of that knowledge structure.

The Student Model module is made up conceptually of the four components shown in Figure 3, and which are described below:

1. The domain hierarchy, which relates all of the groups (rules), concepts, and knowledge and skill elements
2. The instantiated domain hierarchy which is used to track the state of the student's mastery over the domain; this instantiation is loosely referred to as the Student Model and contains student-specific information
3. The collection of methods which create/instantiate and initialize this knowledge structure for a new student
4. The set of methods which update the internal states of the knowledge structure when new information concerning the student's knowledge is obtained from running a scenario/sub-scenario or after the Tutoring Strategy module has remediated the student



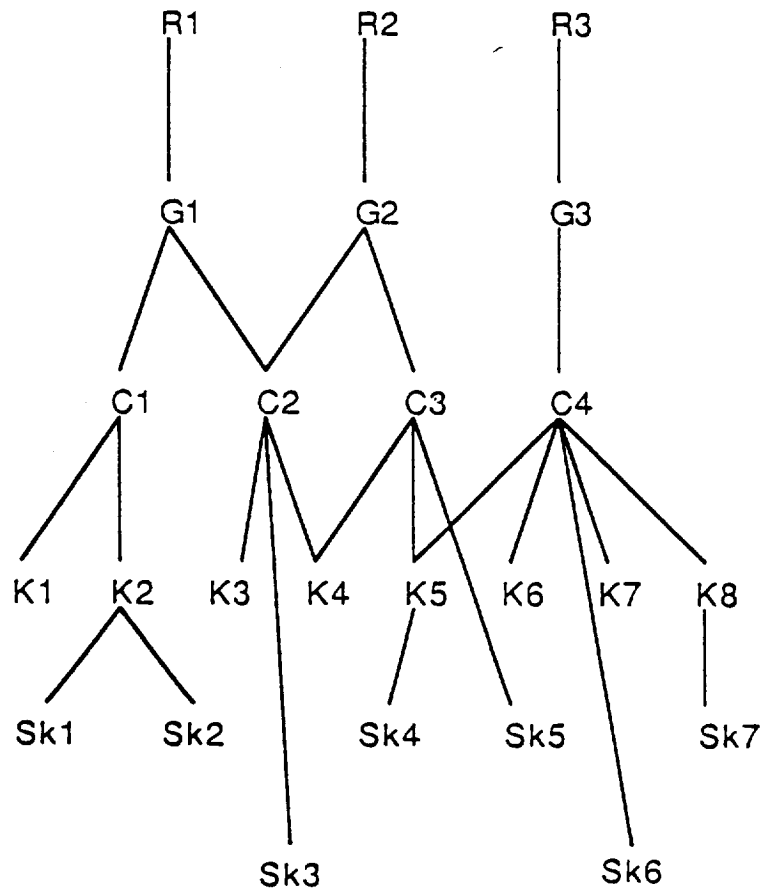
**Figure 3. The Student Model module is a collection of knowledge structures and methods for maintaining the internal states of the knowledge structures**

## 2. REPRESENTING THE DOMAIN HIERARCHY

The domain hierarchy is the set of instructional concepts that must be mastered by the student. The domain hierarchy is developed from an analysis of the skills and concepts needed to successfully manipulate the RMS.

### 2.1 The Domain Hierarchy is Represented as a Network of Nodes

The domain hierarchy can be represented as a network of rules or groups, concepts, and knowledge/skill elements as shown in Figure 4.



Note:

R1 - 3 = Rules

G1 - 3 = Groups

C1 - 4 = Concepts

K1 - 8 = Knowledge Elements

SK1 - 7 = Skill Elements

Figure 4. Network of groups, concepts, and knowledge are linked to the results of Performance and Error Analysis.

Each item of the network shown in Figure 4 (e.g., R1, C3, K10) is a node of that network. A node contains information concerning concepts which require mastery. In other words, the node *represents* the student's knowledge about how to apply the underlying concepts (e.g., ranking and ordering of possible actions). Therefore, in our discussions on the content of the nodes, we will make no reference to the conceptual content.

A node in the domain hierarchy contains three basic attributes (more will be added later in this discussion). These attributes are:

1. The type of node (Rule, Group, Concept, or Knowledge/Skill Element)
2. The set of its parent nodes
3. The set of its children nodes

For example, from Figure 4, the parent node of Group 2 is Rule 2. The children nodes of Group 2 are Concept 2 and Concept 3.

## 2.2 Representing the Domain Hierarchy Network

We will use the object-oriented language "C++" to represent the domain hierarchy. Each node in the domain hierarchy will be represented as a instantiation of a domain hierarchy node class. The interconnections of the domain hierarchy will be represented by pointers to objects. The generic object for a domain hierarchy will look something like this:

```
domain_node {  
    node_type           // to store the node type  
  
    node_parents        // a list of pointers to the nodes that  
                        // are the immediate ancestors of the node  
  
    node_children       // a list of pointers to the nodes that  
                        // are the immediate descendants of the  
                        // node  
}
```

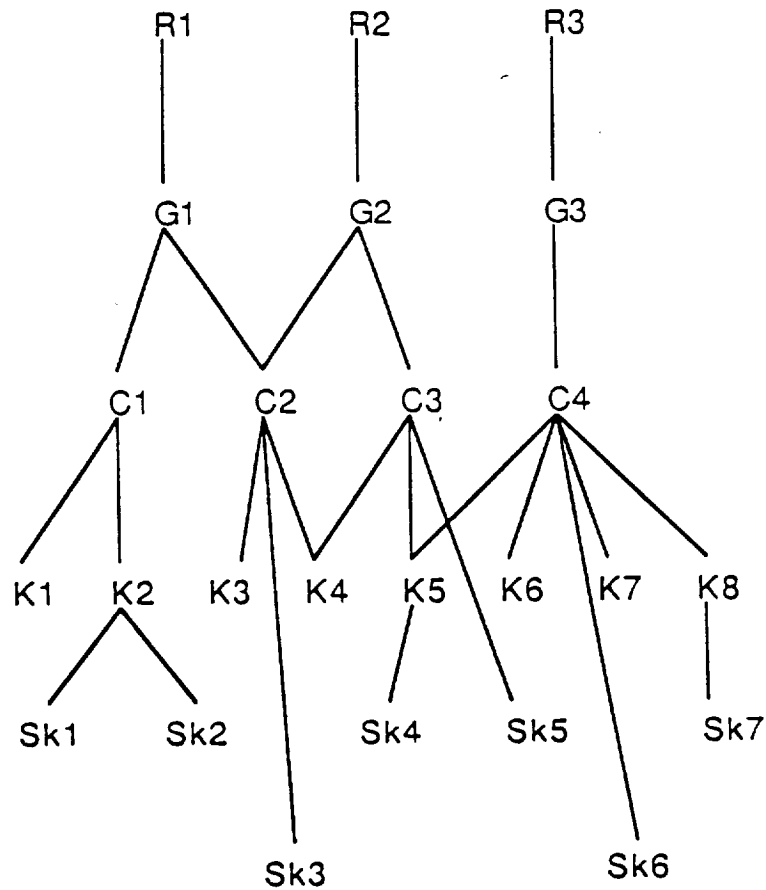
## 2.3 Student Model is a Virtual Copy of the Domain Hierarchy

The domain hierarchy allows us to relate the expert model to a set of concepts that the student must master. The domain hierarchy, by itself, is insufficient to keep track of the performance of the student. To do that requires that we expand this network to consider student performance and training parameters (e.g., for training, which node was remediated).

We begin to create the student model by making a virtual copy of the domain hierarchy (see Figure 5), then modifying it by adding attributes to each of the nodes. The overall concept is shown in Figure 5. The Student Model is unique to the student; each student has his own Student Model.

## 2. Representing the Domain Hierarchy

---



Note:

R1 - 3 = Rules

G1 - 3 = Groups

C1 - 4 = Concepts

K1 - 8 = Knowledge Elements

SK1 - 7 = Skill Elements

**Figure 5. The Student Model is a copy of the domain hierarchy**

#### 2.4 Additional Attributes of the Student Model

A new instantiation of the domain node class is created for each of the nodes of our Student Model. They include the values inherited from the domain hierarchy nodes plus the following new data:

- a. Current State
- b. Last State
- c. Times Used
- d. Last Used Date/Time
- e. Times Mastered
- f. Last Mastered Date/Time
- g. Times Misused
- h. Last Misused Date/Time
- i. Time Remediated
- j. Last Remediated Date/Time
- k. Decay Rate

These attributes are explained in detail in the following sections. The Instructional Assignment also maintains information in each node in the Student Model. See the "P2T2/ITS Instructional Assignment Design" for an explanation of the data that the Instruction Assignment module uses.

For the general class node in the Student Model network the data object would look like:

```
domain_node {  
    // domain hierarchy structure information  
    node_type          // to store the node type  
  
    node_parents       // a list of pointers to the nodes that  
                      // are the immediate ancestors of the node  
  
    node_children      // a list of pointers to the nodes that  
                      // are the immediate descendants of the  
                      // node  
  
    // student data  
    current_state      // current node state  
  
    previous_state     // previous node state
```



## 2. Representing the Domain Hierarchy

---

```
times_used          // the number of times the skill or
                    // concept has been employed by the student

last_used           // records when the node was last used

times_mastered      // the number of times the student has
                    // demonstrated mastery of the skill or
                    // concept

last_mastered       // records when the node was last mastered used

times_misused       // the number of times the student has
                    // misused the skill or concept

last_misuse         // when the student last misused the
                    // skill or concept

times_remediated    // the number of times the student has
                    // received remediation on the skill or
                    // concept

last_remediated     // when the student was last remediated
                    // on the skill or concept

}
```

These nodes are described in the following segments. (For data specifically related to the Instruction Assignment module, see "P2T2/ITS Instructional Assignment Design".)

### a. Current State

While training on the system, the student will either show mastery of a particular concept or make an error of some type. We keep track of the student's successes and failures in the Student Model. In particular, the `current_state` attribute is used to keep track of the student's mastery of the node. We define five possible states:

**MASTERED** - the student has demonstrated a complete understanding of the group, concept, or knowledge element.

**UNKNOWN** - the ITS knows nothing about the student's performance because s/he has not been tested yet or the ITS has reset the state to UNKNOWN.

**SUSPECT** - the ITS suspects that this node is a cause of the student's misconception.

**MISUSED** - the student has misused a rule, concept, knowledge element, or skill.

**ERROR** - the ITS has determined that this node is the actual source of the student's misconception or insufficient skill.

**b. Last State**

This slot contains the status of the node during the previous mastery cycle just prior to the isolation of the error node(s). A mastery cycle is defined to be a collection of scenarios or remediations leading to the complete mastery of the domain. Once mastery is achieved, the ITS resets the current states of all nodes.

**c. Times Used**

Every time the node is used during the course of running a part-task, this attribute value is incremented by 1.

**d. Last Used Date/Time**

This slot contains the combination date and time that the student used the node.

**e. Times Mastered**

Every time the student is successfully tested and this particular node is involved the times\_\_mastered attribute is incremented by one.

**f. Last Mastered Date/Time**

Every time the student demonstrates mastery over the node, the current date and time is recorded in the last\_\_mastered attribute.

**g. Times Misused**

Every time the student is tested, this particular node is involved, and the student makes an error (misuses the node), the times\_\_misused attribute is incremented by one. The times\_\_misused attribute therefore contains the total number of times that the student made an error involving the node.

**h. Last Misused Date/Time**

Every time the student misuses the node, the current date and time is recorded by the last\_\_misuse attribute.

**i. Times Remediated**

Every time the student is remediated on this particular node, the times\_\_remediated attribute is incremented by one.

**j. Last Remediated Date/Time**

Every time the student is remediated on the node domain, the current date and time is placed in the last\_\_remediated attribute.

**k. Decay Rate**

This slot contains the average time that it takes for the student to lose mastery over the node and to misuse it in some way.

This network of node objects represents a portion of the Student Model. The Student Model contains other types of information, but these are not discussed in this report since they are not directly related to the Student Model module.

### **3. STUDENT MODEL INITIALIZATION**

Student Model initialization refers to the processes associated with initially creating a student model, seeding it with knowledge about the student's mastery over basic concepts, then maintaining a representation of the student's overall mastery of the domain hierarchy.

#### **3.1 New Student Model Creation**

The initialization process (shown in Figure 6) consists of two steps:

- (1) Determine if a Student Model exists for a specific student
- (2) For new students, present and evaluate a pre-test to populate the Student Model. Make a virtual copy of the domain hierarchy and add new schema slots.

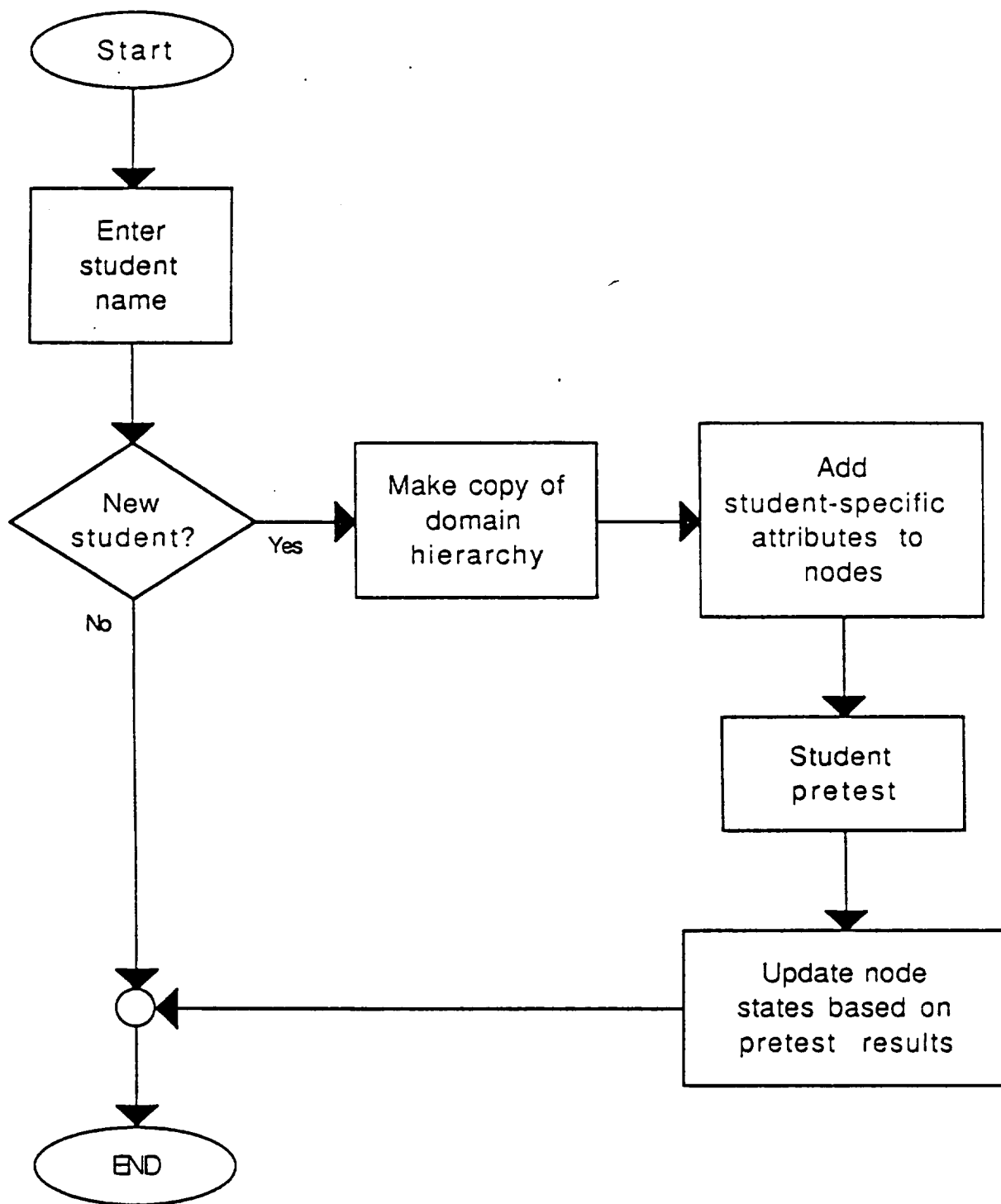


Figure 6. New student module initialization process

#### 3.1.1 Determine Whether Student Is New or Continuing

To begin a training session, the student logs onto P2T2/ITS by typing his/her name. If the system cannot find a stored student model, then the system asks him if s/he is a NEW STUDENT. If s/he answers no, then the system loads the file. A student with a Student Model file is called a CONTINUING STUDENT.

#### 3.1.2 Handling New Students

For new students, the system must create a Student Model file and initialize it appropriately. The first step in this process is to make a virtual copy of the domain hierarchy network, and then modify the node schemata appropriately by adding slots and inserting initial slot values into them. This process is shown in Figure 7 below.

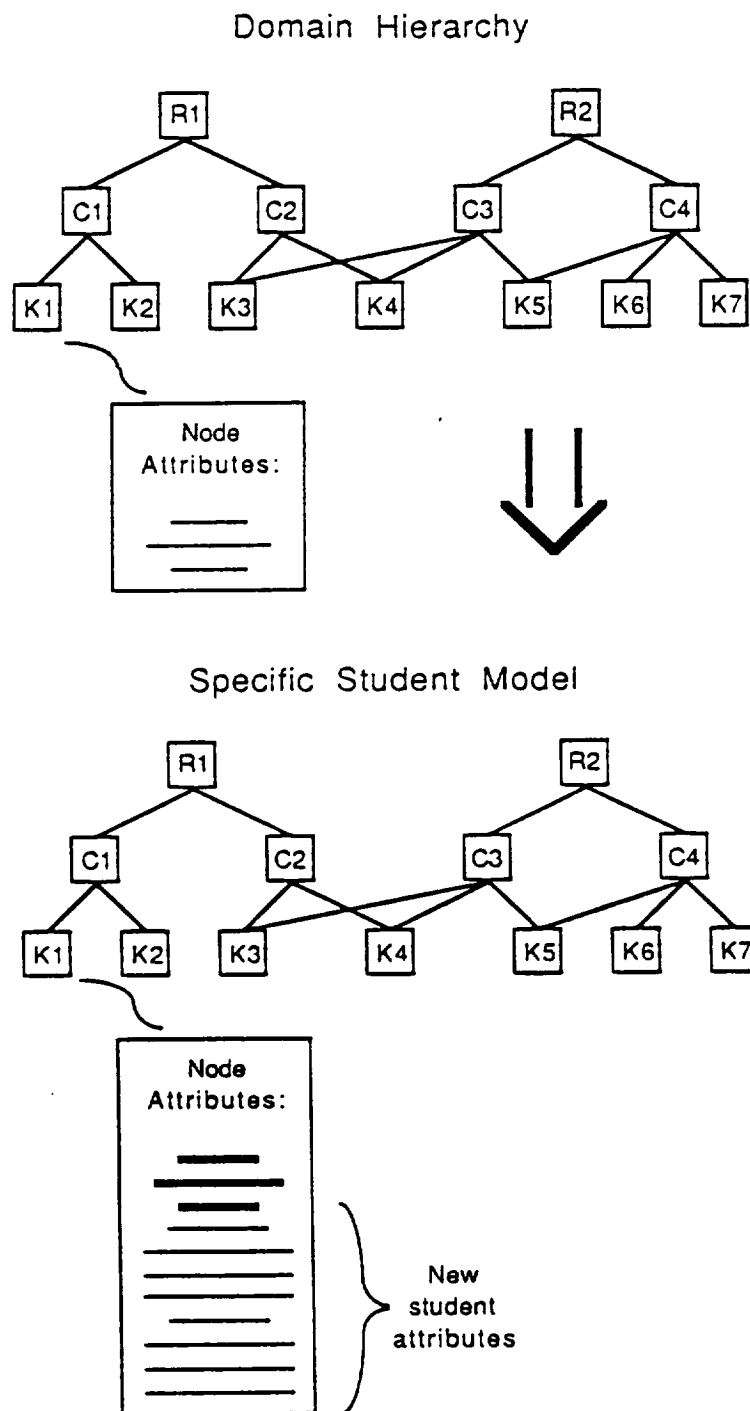


Figure 7. Building a student model from the domain hierarchy

The initial slot values (or default values) for a new student node are:

```
domain_node {  
    // student data  
    current_state = UNKNOWN  
  
    previous_state = UNKNOWN  
  
    times_used = 0  
  
    last_used = NULL  
  
    times_mastered = 0  
  
    last_mastered = NULL  
  
    times_misused = 0  
  
    last_misuse = NULL  
  
    times_remediated = 0  
  
    last_remediated = NULL  
}
```

### 3.2 Pre-test Student and Update Student Model

To get an initial feel for the knowledge or skill level of the student, we begin by asking a series of questions concerning the knowledge elements of the domain hierarchy. As each question is answered correctly or not, the particular knowledge element node has its object attributes updated. Once the Student Model has been updated based on the series of questions posed, the new student then has all of the features of a continuing student and is treated as such from then on. The Student Model file is stored to disk at this time.



#### 4. STUDENT MODEL NODE STATE PROPAGATION

The Student Model Module is both a copy and an enhancement of the domain hierarchy (i.e., a set of nodes) and a set of rules or functions for propagating and updating node states (i.e., the current state attribute of each Student Model node). This section describes the functionality associated with the propagation of node states in the Student Model.

As mentioned before, each node of the Student Model has an attribute called the current state. This value represents the system's knowledge about the student's mastery of that node. The current state attribute has the allowable values defined in Table 1. (The terms "current state" and "state" are synonymous in this report.)

Whenever the state of a particular node (called the "Active Node") is set to a new value by the Error Analysis module, that state may cause the state of its children nodes or its parent's nodes to change as well. This change process is accomplished via a mechanism called "node state propagation" or simply "propagation." States can be propagated downward through the Student Model nodal network toward the knowledge and skill elements, or in certain instances, propagated upward toward the parent nodes. Beginning with a node which has recently had its current state attribute changed, we first perform an in-order traversal of the Student Model node network and propagate these state changes downward. We then determine if there is a need to propagate states upward as well. Finally, we review the state of each node to see if additional state changes are possible based the relationship of each node to its children.

In the subsections to follow, "node propagation" is described and presented pictorially. Figure 8 provides the standard symbology and notation for describing nodes, nodal relationships, node states, and propagation directions. Section 4.1 describes the downward propagation process. Section 4.2 describes the upward process. Section 4.3 describes the state changes motivated by the relationships to the child node states.

Table 1. Allowable Student Model node states

STATE	DESCRIPTION
UNKNOWN	Nothing is known about the student's mastery of this node.
MISUSED	In running a scenario or sub-scenario, the Performance Analysis and Error Analysis modules showed that the student misused the concepts, knowledge elements or skills associated with the node (analogous to testing a signal of a particular component and discovering that it is BAD or out-of-tolerance).
MASTERED	In running a part task or whole task, the Performance Analysis and Error Analysis modules indicated or implied that the student successfully applied knowledge or skills associated with that node (analogous to testing a signal of a particular component and discovering that it is GOOD or in tolerance).
ERROR	NOT to be confused with MISUSED! A node in this state means that the student has MISUSED the knowledge or skills associated with the node because of a fundamental misunderstanding or lack of skill and NOT because he is lacking lower-level skills or knowledge represented by lower-level nodes of the Student Model.
SUSPECT	A suspect node has a "suspicion value" associated with it. The larger the suspicion value, the higher the suspicion (or likelihood) of that node actually being an ERROR node. (This determination is eventually made through a converging set of tests.) In discussions which follow, we use the notation "SUSPECT+1" to mean that in setting the node to a higher state of suspicion we increment its suspicion value by one. Therefore, a node with a suspicion value of 2, when reset, will have the value "2 + 1" or simply, "3."

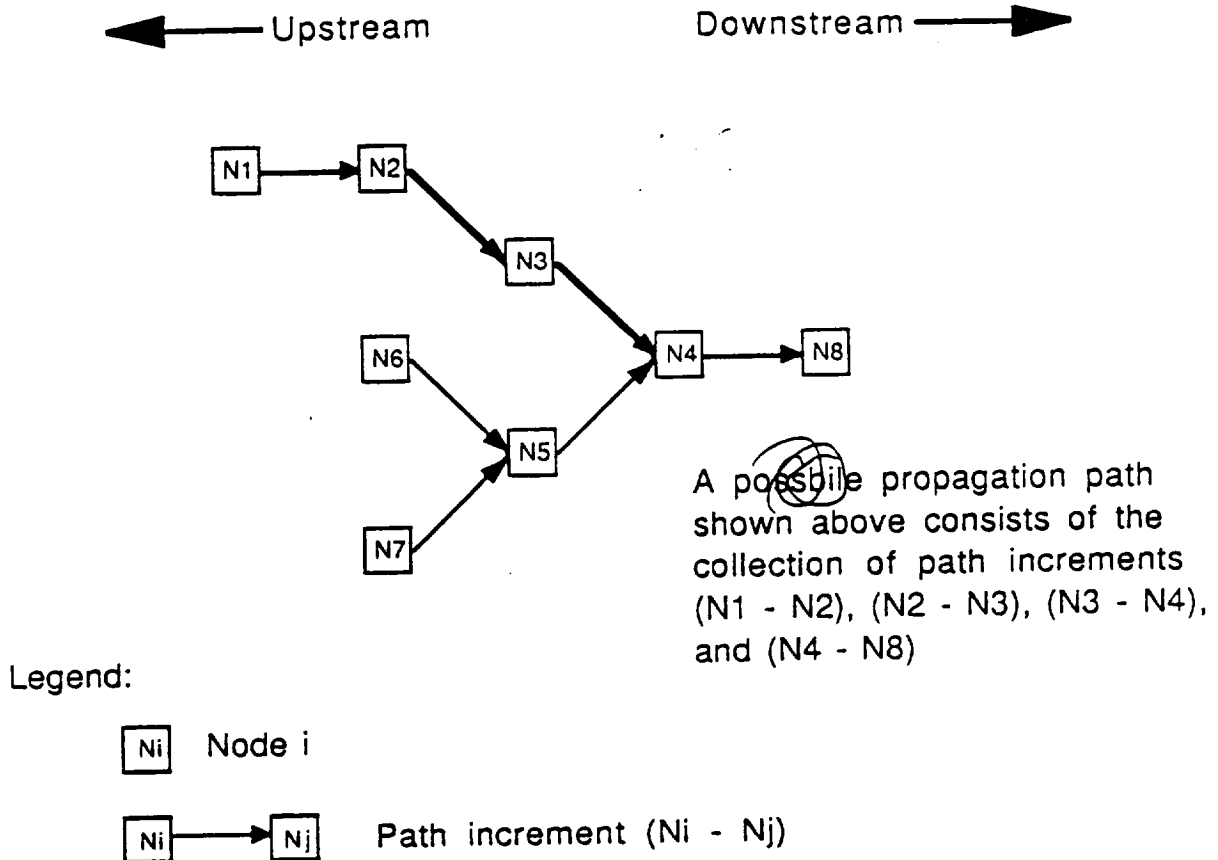


Figure 8. Standard symbology used in representing node state propagation

#### 4. Student Model Node State Propagation

##### 4.1 Downward Propagation of Student Model Node States

The matrix in Table 2 shows how to set the new state of an affected node based on the parent's new state and the node's current state.

Table 2. Student model node state down-stream propagation matrix

Parent Node's Current State	Child Node's New State				
	Mastered	Unknown	Suspect	Misused	Error
Mastered	<del>Mastered</del> None	<del>Mastered</del> None	<del>Mastered</del> None	<del>Mastered</del> None (Mastered or more)	None
Unknown	<del>Mastered</del> None	None	<del>Suspect+I</del> None	<del>Suspect+I</del> None	None
Suspect	Mastered	<del>Suspect</del> None	Suspect+I	Suspect+I + misused	None
Misused	<del>Misused</del> misused or None	<del>None</del> Suspect	<del>Suspect</del> None	<del>Misused</del> None Suspect	None
Error	None	None	None	None	None

Notes:

\* Logical Inconsistency

None No State Change, Do not continue propagating beyond this node

Several examples will help in understanding the above matrix. Suppose the active node has a current state value of MASTERED. Also suppose that the active node has one child node whose current state is UNKNOWN. Per the matrix above, the new state of the child node will be MASTERED. We would then determine if the child node has one or more child nodes of its own. Let's say that it does and that its child also has a current state of UNKNOWN. We would set that child node to MASTERED as well.

Another case, represented by the term "None" in the matrix, is as follows. Suppose that the active node is in the new state, MASTERED, and that its one and only child node is also in the current state MASTERED. For this instance, we not only do nothing about changing the child's state, but we stop propagating states downward to the children of that node. Whenever a "None" is encountered, no further propagation is allowed along that propagation path.

Four examples are provided below for downward node state propagation along various paths. A propagation path is the collection of arcs between dependent nodes of the Student Model. Each example represents a specific type of propagation problem and is

#### 4. Student Model Node State Propagation

---

intended to represent only a subset of an actual Student Model. In each example, we show how the states of the Student Model nodes change through propagation after a particular node state has been changed by the action of the Error Analysis module.

In the first example (see Figure 9), we have the simplest case of a node which is a "terminal node"; that is, the node represents an atomic (non-divisible) knowledge element or skill. If the system detects that the student misused a terminal node, then there is no propagation required.

In the second example (see Figure 10), we have the case of a node which is dependent upon a linear chain of other nodes. In this case, the node states are propagated from the active node back toward the terminal node.

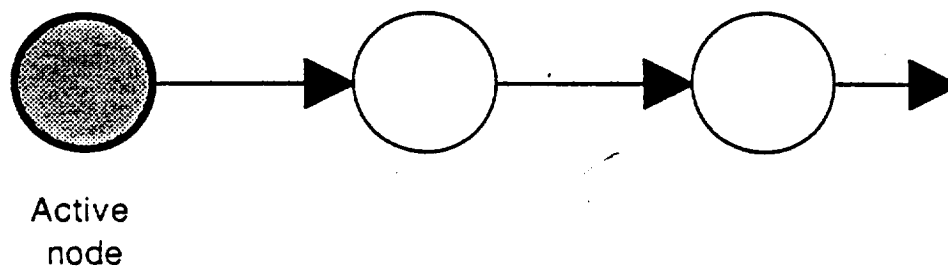


Figure 9. Downward node propagation at a terminal node

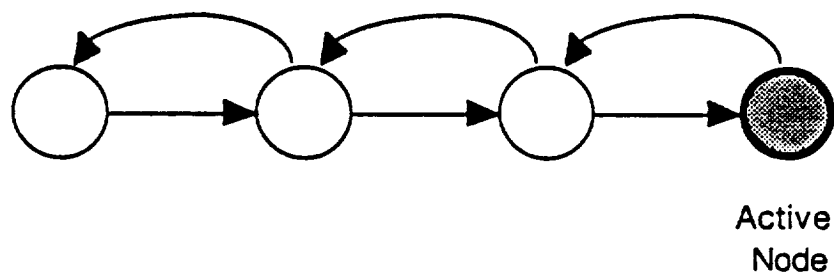
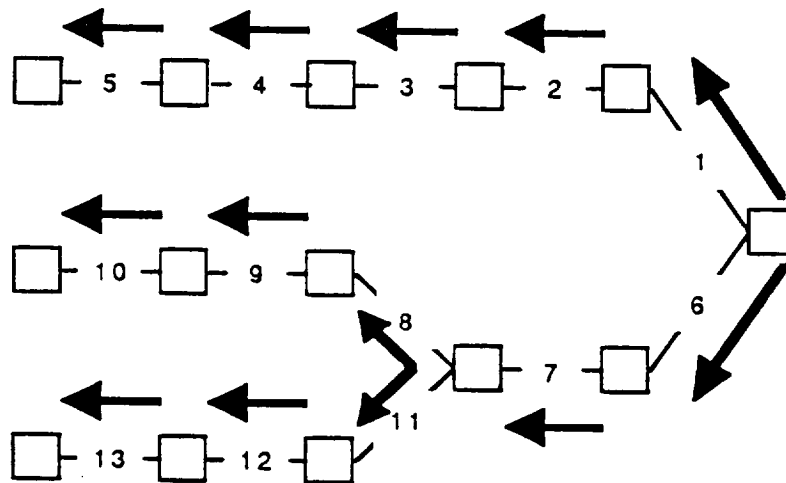


Figure 10. Downward node propagation through a linear chain of nodes

#### 4. Student Model Node State Propagation

In the third example (see Figure 11), we have a case in which there is more than one dependent pathway. This example also represents the case where the multiple pathways are completely parallel and independent. The node state propagation proceeds from the active node to the first junction, then (conceptually) splits into two or more propagation streams.



**Figure 11. Downward node propagation through multiple parallel paths**

#### 4. Student Model Node State Propagation

In the fourth example (see Figure 12), we have the case in which multiple pathways exist and intersect at more than one location. In this case, node state propagation proceeds from the active node to the first junction, then splits into two or more propagation streams as in the previous example. However, at the point where the streams intersect again, only one stream is allowed to pass. In other words, we do NOT allow two propagation streams because this could double the suspicion value of certain nodes incorrectly.

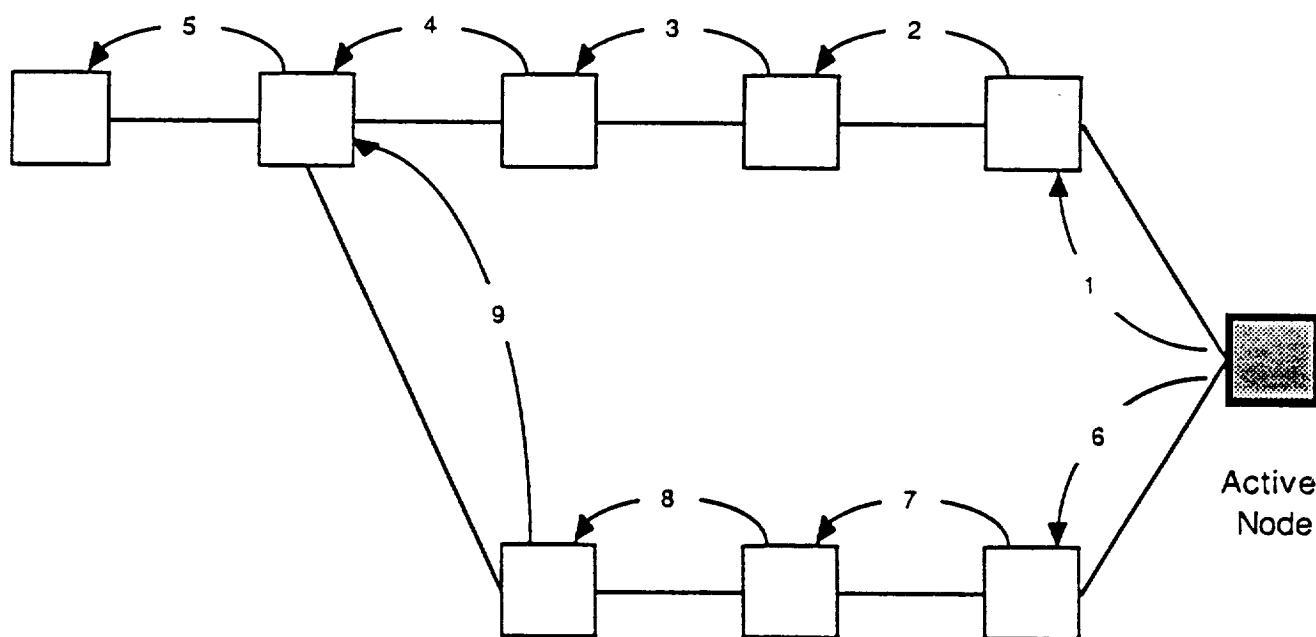


Figure 12. Downward node propagation through multiple intersecting paths



#### 4. Student Model Node State Propagation

---

The downward node state propagation matrix of Table 1 can be translated into the following propagation rules:

**Rule #1**

```
IF current_state = MASTERED
AND child node's current_state = UNKNOWN
THEN
SET child node's current_state to MASTERED
```

**Rule #2**

```
IF current_state = MASTERED
AND child node's current_state = SUSPECT
THEN
SET child node's current_state to MASTERED
```

**Rule #3**

```
IF current_state = SUSPECT
AND child node's current_state = UNKNOWN
THEN
SET child node's current_state to SUSPECT
SET child node's suspicion_value to 1
```

**Rule #4**

```
IF current_state = SUSPECT
AND child node's current_state = SUSPECT
THEN
INCREMENT child node's suspicion_value
```

**Rule #5**

```
IF current_state = MISUSED
AND child node's current_state = UNKNOWN
THEN
SET child node's current_state to SUSPECT
SET child node's suspicion_value to 1
```

**Rule #6**

```
IF current_state = MISUSED
AND child node's current_state = SUSPECT
THEN
INCREMENT child node's suspicion_value
```

#### 4.2 Upward Propagation of Student Model Node States

Upward propagation of Student Model node states is only done when the state of an active node has just been set to MISUSED. Upward propagation is used to set the state of

#### 4. Student Model Node State Propagation

upstream nodes that are either unknown or suspect to MISUSED. This is done because, by definition, any parent node is dependent upon the state of its children. If one or more of the children have been MISUSED, there is no way that a higher level construct (i.e., the parent node) could be logically performed properly.

Upstream propagation can be summarized by the matrix in Table 3.

**Table 3. Student model node state upstream propagation matrix**

Child Node's Current State	Parent Node's New State				
	Mastered	Unknown	Suspect	Misused	Error
Mastered	None	None	None	None	None
Unknown	None	None	Misused	Misused	None
Suspect	None	None	Misused	Misused	None
Misused	None	None	None	None	None
Error	None	None	None	None	None

**Notes:**

None      No State Change, Do not continue propagating beyond this node

The rules which implement the matrix of Table 3 above are summarized below:

**Rule #1**

IF current\_state = SUSPECT  
AND child\_node's current\_state = UNKNOWN  
THEN  
SET current\_state = MISUSED

**Rule #2**

IF current\_state = SUSPECT  
AND child\_node's current\_state = SUSPECT  
THEN  
SET current\_state = MISUSED

#### 4.3 Evaluating Node State Relationships to Determine Actual Student Errors

After downward and upward propagation are completed, each node is investigated to see if an actual student error has been found. A student error is represented by a node with a current state value of ERROR. There are two ways in which a node can be given a current state value of ERROR.

- (1) If the node is a terminal node and the current state of the node is MISUSED, or
- (2) If the node is not a terminal node and its current state is MISUSED and all of its child nodes are in state, MASTERED.

These two criteria are represented by the two rules:

##### Rule #1

IF the node is a terminal node  
AND current\_\_state = MISUSED  
THEN  
SET current\_\_state = ERROR

##### Rule #2

IF the node is not a terminal node  
AND EACH child node's current\_\_state = MASTERED  
THEN  
SET current\_\_state = ERROR

#### 4.4 Examples of Full Node State Propagation

In this section, we provide an example of how "node states" are propagated through the Student Model. For this example, we shall ignore the workings of the other P2T2/ITS modules; we are only concerned here with the states of the nodes themselves. We assume that the student runs through a part-task and that, as a result, one or more node states are set to either the value MISUSED or MASTERED by the Error Analysis module. We also assume that the system knows how to select test nodes. (This subject is discussed in the Instructional Assignment design.)

##### 4.4.1 Node State Propagation Through a Branch

Given the Student Model structure shown in Figure 13 we begin with Test 1 (T1) having set node 8 to the state MISUSED. Applying downward propagation, nodes 1 through 7 are set to SUSPECT with an integer value of 1. Upward propagation yields nothing because node 8 has no parent nodes.

The system decides to test node 6 next. (See the "NASA P2T2 Intelligent Trainer Deficiencies Analysis Design" for a description of the node test selection algorithm.)

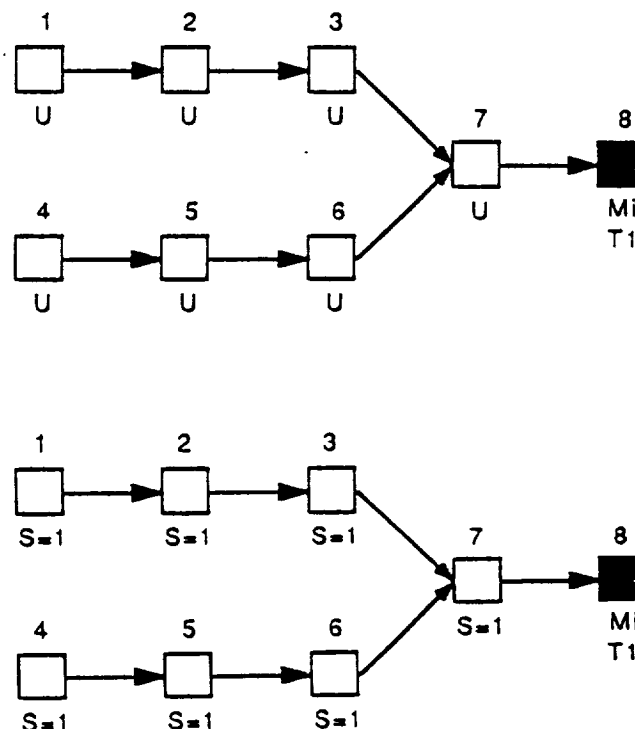


Figure 13. (Part 1 of 4) Example branching propagation

#### 4. Student Model Node State Propagation

The result of the second test (T2) is that active node 6 has been MASTERED. Figure 14 shows this state before and after signal propagation. Downward propagation sets nodes 4 and 5 to MASTERED. No upward propagation is attempted.

The system next recommends node 3 for testing.

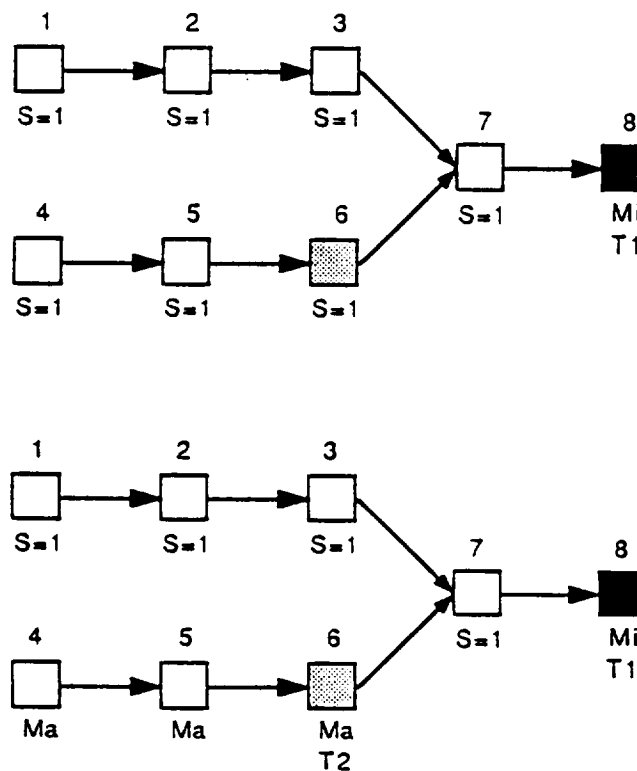


Figure 14. (Part 2 of 4) Example branching propagation

#### 4. Student Model Node State Propagation

Figure 15 (part 3) shows that the result of test 3 (T3) was that node 3 was MASTERED. Applying downward propagation, nodes 1 and 2 are set to MASTERED as well. Again, no upward propagation is attempted.

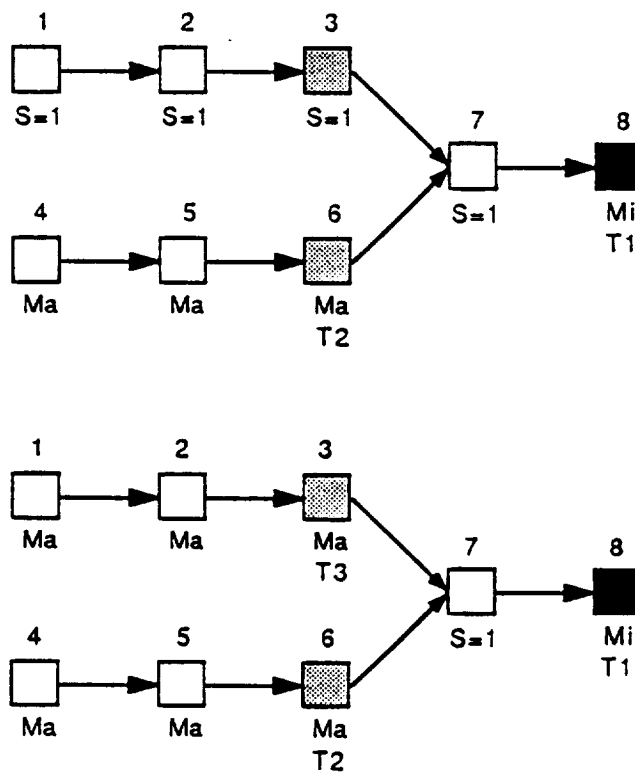


Figure 15. (Part 3 of 4) Example branching propagation

#### 4. Student Model Node State Propagation

The system next recommends node 7 for testing.

Figure 16 (part 4) shows that the result of test 4 (T4) was that node 7 was MISUSED. Applying downward propagation yields no state changes. Neither does applying upward propagation.

Evaluating the state of each node, we discover that node 7 is in the state MISUSED and both of its child nodes, 3 and 6, are MASTERED. Therefore, node 7 is reset to state ERROR.

The system recommends that node 7 be remediated.

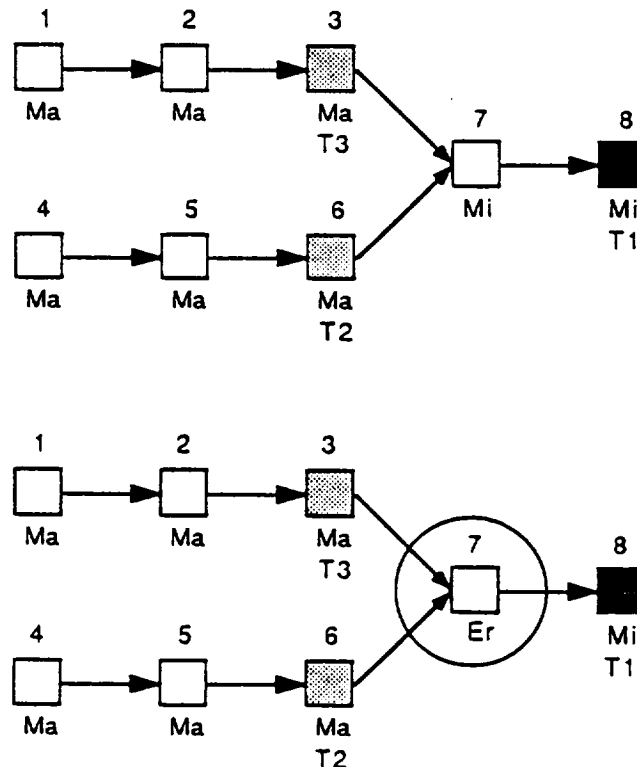


Figure 16. (Part 4 of 4) Example branching propagation





<b>APPENDIX J: DEFICIENCIES ANALYSIS DESIGN</b>
---



Section	Section Description	Page
1.	INTRODUCTION .....	5
1.1	Basic Assumptions.....	5
1.2	Existence of the Domain Hierarchy.....	5
1.3	Functions of the Student Deficiency Module.....	7
2.	RELATIONSHIP TO OTHER P2T2/IT MODULES.....	8
2.1	Initializing the Student Model.....	8
2.2	Diagnosing Student Deficiencies.....	8
3.	TESTING OR REMEDIATION.....	9
4.	DETERMINING THE STUDENT MODEL NODE TO TEST.....	11
4.1	Defining the Suspect Group .....	11
4.2	Reducing the Suspect Group .....	12
4.3	Calculating Relative Probability of Failure.....	12
4.4	Assigning Node Test Values .....	13
4.4.1	Calculating ElimMastered(s).....	14
4.4.2	Calculating ElimMisused(s).....	15
4.5	Test Cost Analysis.....	16
4.5.1	Calculating ProbMisused(s) .....	17
4.6	Final Selection .....	18

GLOBAL INFORMATION SYSTEMS TECHNOLOGY, INC.  
NASA P2T2 Intelligent Trainer Design  
Deficiencies Analysis Design Document  
August 17, 1990

List of Tables

Table	Table Description	Page
1.	Calculating $1/MTBD_i$ Terms.....	13
2.	Listing of $FC_i$ Terms.....	13
3.	Calculating ElimAve(s) Terms.....	16
4.	Listing of ElimAve(s) Terms.....	18
5.	Ranked listing of Nodes.....	19

Figure	Figure Description	Page
1.	Re-orienting the student model.....	6
2.	Overview of P2T2/IT architecture. ....	8

## 1. INTRODUCTION

This document describes the Deficiencies Analysis module of the NASA P2T2 Intelligent Trainer. For an introduction to the functions of the other modules, refer to the following documents describing other modules and aspects of P2T2/IT:

- P2T2 Intelligent Trainer Design Overview
- P2T2 Intelligent Trainer Performance Analysis Design
- P2T2 Intelligent Trainer Error Analysis Design
- P2T2 Intelligent Trainer Student Model Design
- P2T2 Intelligent Instructional Assignment Design
- Modifications to P2T2 Design

### 1.1 Basic Assumptions

The design of the Deficiencies Analysis module is based on a set of assumptions about other P2T2/IT modules. These assumptions are described in sections 1.2 and 1.3.

### 1.2 Existence of the Domain Hierarchy

The first assumption is that there is an entity called the "*domain hierarchy*" which is the network of nodes representing the groups, concepts, sub-concepts, knowledge elements, and skill elements derived from the domain of the Remote Manipulator System (RMS). This domain hierarchy is represented using C++ classes. The contents of the C++ classes are presented in Section 2 of the "NASA P2T2/IT Student Model Design."

From this domain hierarchy, a "student model" is created in which the states of the student's mastery over the domain are represented and maintained.

In representing the domain hierarchy and student model previously in the "NASA P2T2/IT Student Model Design," we depicted the network of nodes as a tree structure with the higher-level nodes shown at the top and the lower-level ones at the bottom. For greater clarity in this document, we shall at times rotate the tree onto its side and show lower-level nodes on the left and higher-level nodes on the right. This change is shown in Figure 1 below.

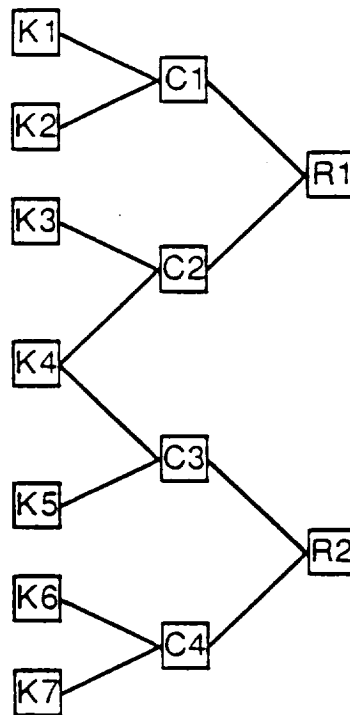
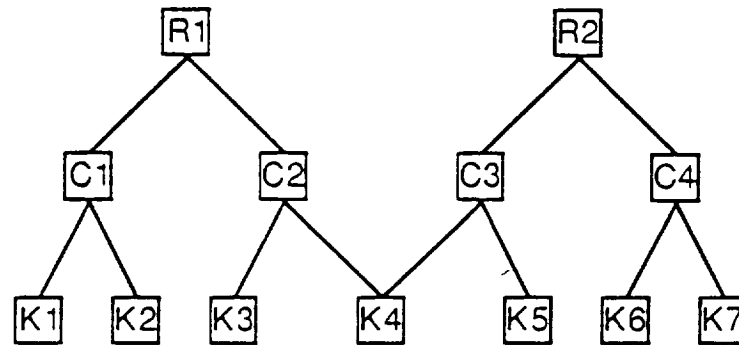


Figure 1. Re-orienting the student model

### **1.3 Functions of the Student Deficiency Module**

The Deficiency Analysis module has three primary functions:

1. Diagnose student weaknesses by indicating which nodes of the student model have been misused or are highly suspect.
2. Develop a plan to confirm the diagnosis through testing (via running a part task or whole task without coaching or other instructional interaction).
3. Recommend a particular set of nodes of the student model for remediation by the Instructional Assignment module



## 2. RELATIONSHIP TO OTHER P2T2/IT MODULES

The Deficiencies Analysis module operates on information contained in the Student Model. It decides whether to test for a suspected student weakness or to recommend remediation on a concept, knowledge element, or skill. The recommended test or remediation node is passed to the Instructional Assignment module for further action.

Figure 2 shows the overall functional flow of the system through modified P2T2, Performance Analysis, Error Analysis, Student Model, Deficiencies Analysis, and Instructional Assignment.

### 2.1 Initializing the Student Model

When a student first begins using the system, the Student Model module is responsible for identifying the individual, then creating/instantiating a unique student model for him/her. For continuing students, the Student Model module is responsible for accessing and updating the internal states of the student model in response to new information revealed through running a part task or whole task. The primary mechanism used is node state propagation described in the "NASA P2T2/IT Student Model Design."

### 2.2 Diagnosing Student Deficiencies

The diagnosis phase consists of evaluating the student model and determining where the student has shown knowledge and skill masteries or deficiencies. If the student has no present weakness, P2T2/IT will create and present part tasks or whole tasks to keep his or her skills current. The diagnosis function is responsible for four items:

- (1) Detecting where potential weaknesses lie,
- (2) Making reasonable hypotheses about what the actual student misconceptions or skill deficiencies are, or conversely, determining areas of mastery,
- (3) Recommending the presentation of some whole task or part task (the parameters of which are determined by the Instructional Assignment module) to confirm the hypotheses, and
- (4) Recommending the remediation of individual or combined skill and knowledge elements to the Instructional Assignment module.

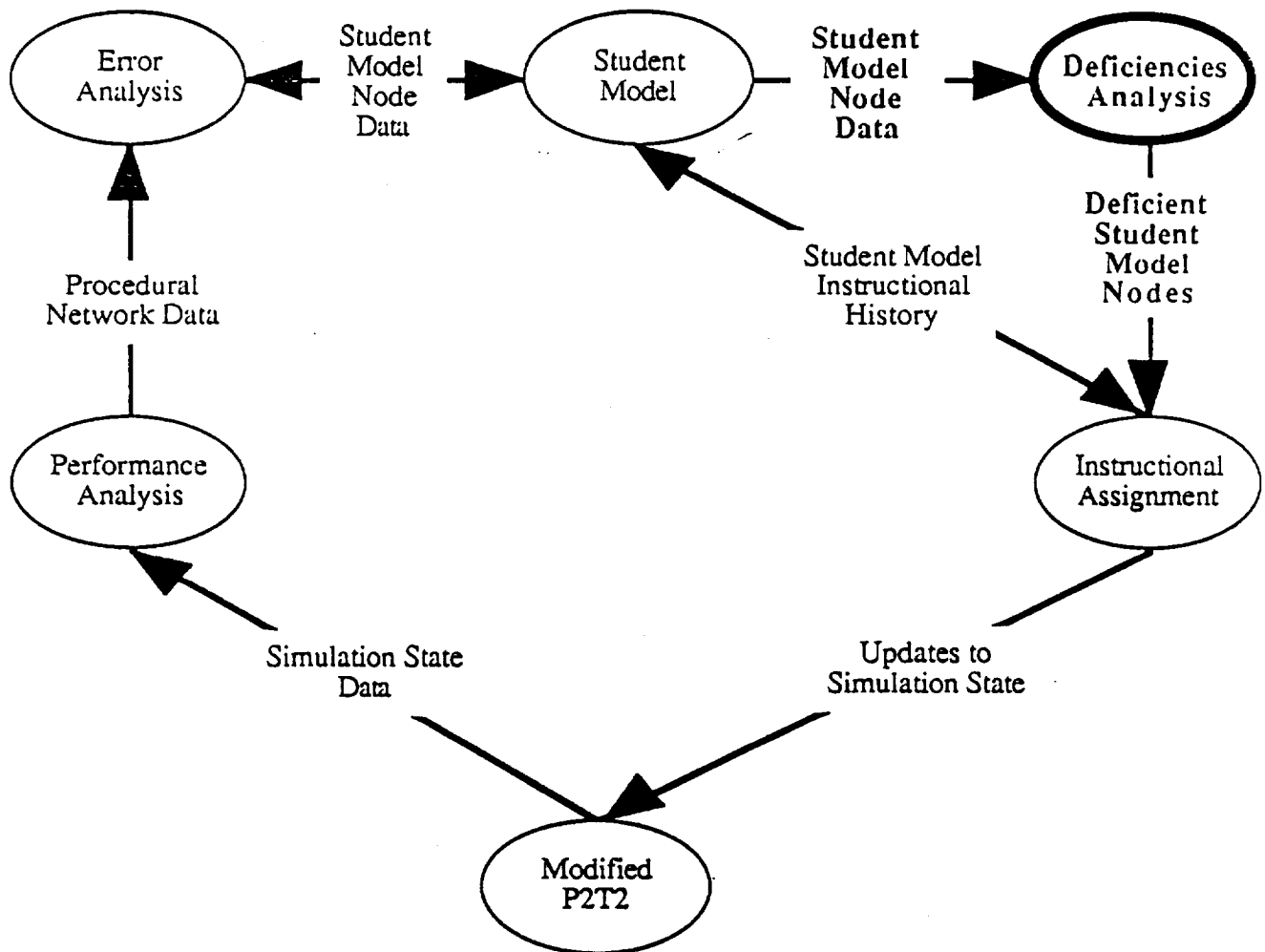


Figure 2. Overview of P2T2/IT architecture.

### 3. TESTING OR REMEDIATION

The decision to test or remediate is a simple one based on reviewing the states of all of the nodes in the student model. This topic is covered in detail in the "NASA P2T2/IT Student Model Design". In essence, we look at all nodes and make a list of those nodes that have a current state value of ERROR. If there are any such nodes, then the decision is to remediate. If there aren't any nodes in an ERROR state, then the decision is to test. Which node to test is determined by following the process described in section 4 of this document.

When the decision has been made to remediate, control is passed to the Instructional Assignment module, which has the responsibility of providing guidance and practice to help the student overcome his or her determined conceptual errors or skill deficiencies. When the Instructional Assignment module has completed its work, control is passed back to the Deficiencies Analysis module, which then has the task of confirming that the student has indeed mastered the particular nodes. The states of the Student Model must be reset appropriately to allow the system to evaluate the student's performance under testing conditions (i.e., by running a whole task or part task to confirm that the student has mastered the node).

#### 4. DETERMINING THE STUDENT MODEL NODE TO TEST

One of the goals of the Deficiencies Analysis module is to select a series of nodes to test by presenting the student with part task or whole task which, when taken over a large enough sample of sessions, generates appropriate data to isolate a true student misconception or skill deficiency (i.e., an ERROR node). The algorithm responsible for this "fault isolation diagnosis" selects one node per part task/whole task iteration that eliminates the greatest amount of relative probability of failure per unit time. (The term "relative probability of failure" is defined in Section 4.3.)

The algorithm for determining the optimal node to test is divided into six main steps, which are discussed in the remainder of this document:

- 4.1 Defining the Suspect Group
- 4.2 Reducing the Suspect Group
- 4.3 Calculating Relative Probability of Failure
- 4.4 Assigning Node Test Values
- 4.5 Test Cost Analysis
- 4.6 Final Selection

##### 4.1 Defining the Suspect Group

The student model is comprised of various types of nodes. At any particular time, these nodes will be in various states. The set of all nodes, regardless of state, is represented by the term, "{A}":

$$\{A\} = \{U\} + \{Ma\} + \{Mi\} + \{E\} + \{S\}$$

where     {U} = the set of all nodes in the state UNKNOWN  
            {Ma} = the set of all nodes in the state MASTERED  
            {Mi} = the set of all nodes in the state MISUSED  
            {E} = the set of all nodes in the state ERROR  
            {S} = the set of all nodes in the state SUSPECT

{S} is also known as the "suspect group."

At each iteration, a list of nodes comprising the suspect group is made or updated. Membership of this group is restricted to nodes whose state is SUSPECT. Nodes in other states may not be included.

##### 4.2 Reducing the Suspect Group

The suspect group may be reduced using the suspicion values themselves.\* If the nodes in the suspect group have varying values of suspicion, we can reduce the suspect group to

---

\* "Suspicion values" were defined as a value expressing the system's belief that the node in question is the source of the student's misconception or error. See specifically section 4.1 (Initializing the Student Model) and 4.2 (Diagnosing Student Weakness) in the "NASA P2T2/IT Student Model Design" for a detailed discussion on generating and propagating suspicion values.

#### 4. Determining the Student Model Node to Test

those nodes that have the *highest* suspicion value. The rationale for doing this is that those nodes with the highest suspicion values represent nodes which can account for most or all of the rules detected as MISUSED by the Error Analysis module.

The final set of nodes that are most suspect is termed the "Most Probable Failing" nodes.

#### 4.3 Calculating Relative Probability of Failure

Definitions:

$FG_i$	the $i$ th node of the Most Probable Failing Group	
$MTBD_i$	Mean Time Between Decay of $FG_i$ , the $i$ th member of the Most Probable Failing Group	
$1/MTBD_i$	is the number of expected misuses of the $i$ th node per unit time	
$FgTot$	Total number of node elements in the Most Probable Failing group	
$TotErrs$	is the sum of $1/MTBD_i$ of each member of the fault group. It is the total number of errors generated per unit time by all members of the fault group.	
$FC_i$	$\frac{1/MTBD_i}{TotErrs}$	is the relative amount of failures contributed by the $i$ th member of the Most Probable Failing Group

Using Table 1 as the example, the number of faults generated per hour for each member of the Most Probable Failing group,  $1/MTBD_i$  is calculated and listed below:

Table 1. Calculating  $1/MTBD_i$  Terms

$i$	$FG_i$	$MTBD_i$	$1/MTBD_i$
1	$E_1$	100	.01
2	$E_2$	100	.01
3	$E_3$	100	.01
4	$E_4$	100	.01
5	$E_5$	100	.01
6	$E_6$	100	.01
7	$E_7$	100	.01
8	$E_8$	100	.01
9	$E_9$	100	.01

The total number in the Most Probable Failing group is 9 (i.e.,  $FgTot = 9$ ).

The total expected errors per unit time, is .09 errors/hour because:

$$\text{TotErrs} = .01 + .01 + .01 + .01 + .01 + .01 + .01 + .01 + .01 = .09$$

The relative amount of failures,  $FC_i$ , contributed by each member of the Most Probable Failing Group is calculated and listed in the table below:

Table 2. Listing of  $FC_i$  Terms

i	$FC_i$
1 .01/.09 =	.111
2	.111
3	.111
4	.111
5	.111
6	.111
7	.111
8	.111
9	.111

#### 4.4 Assigning Node Test Values

We now look at every node in the Student Model to determine which ones could eliminate the largest sum of  $FC_i$  terms. Before doing so, we have to establish the fact that the number of  $FC_i$  terms eliminated depends upon whether the particular node is eventually determined to be MASTERED or MISUSED.

We actually have two terms to consider:

ElimMastered(s) = the total amount of  $FC_i$  eliminated if the node is Mastered.

ElimMisused(s) = the total amount of  $FC_i$  eliminated if the node is Misused.

The probability that a particular node "s" will be Mastered is "ProbMastered(s)", and that the node will be Misused is "ProbMisused(s)".

The amount of  $FC_i$  that is eliminated by testing node "s" during a whole task/part task is, on the average (over a large number of sessions), ElimAve(s).

$$\text{ElimAve}(s) = \text{ProbMastered}(s) \times \text{ElimMastered}(s) + \text{ProbMisused}(s) \times \text{ElimMisused}(s)$$

For any node "s" the outcome of running a part task or whole task will either be that the node was Mastered or Misused. Therefore we have the relationship:

$$\text{ProbMastered}(s) + \text{ProbMisused}(s) = 1$$

#### 4.4.1 Calculating ElimMastered(s)

The term "ElimMastered(s)" is the amount of  $FC_i$  that will be eliminated if the node "s" is Mastered. To calculate this term, we collect the set of nodes that are upstream dependent to node "s" and also include the node "s" in this list. We then create the sum of the  $FC_i$  terms of each node that is both a member of the Most Probable Failing Group and upstream-dependent, and also a member of the defined Most Probable Failing Group.

#### 4.4.2 Calculating ElimMisused(s)

The term "ElimMisused(s)" is the amount of  $FC_i$  that will be eliminated if the node "s" is Misused. To calculate this term, we look at all of the "signal sources"\* upon which the signal source "s" is a dependent input. This is accomplished by the following steps:

1. Create the sum of the  $FC_i$  terms of each signal source that is both a member of the Most Probable Failing Group and downstream-dependent, and also is a member of the defined Most Probable Failing group.
2. Include in the  $FC_i$  eliminated the signal source "s" itself as one of the Most Probable Failing Group members.

---

\* A term from fault-isolation diagnosis, which is where our Deficiencies Model algorithm is taken.

4.5 Test Cost Analysis

Table 3. Calculating ElimAve(s) Terms

Node s	ElimMastered(s)	ElimMisused(s)	ElimAve(s)
1	0.000	1.000	1.000
2	0.111	0.888	0.802
3	0.222	0.777	0.654
4	0.333	0.666	0.555
5	0.444	0.555	0.506
6	0.555	0.444	0.506
7	0.666	0.333	0.555
8	0.777	0.222	0.654
9	0.888	0.111	0.802
10	0.000	0.000	0.000*
11	0.333	0.000	0.333
12	0.333	0.000	0.333
13	0.333	0.000	0.333
14	0.555	0.000	0.555
15	0.555	0.000	0.555
16	0.555	0.000	0.555
17	0.555	0.000	0.555
18	0.555	0.000	0.555
19	0.555	0.000	0.555
20	0.555	0.000	0.555
21	0.555	0.000	0.555
22	0.333	0.000	0.333
23	0.333	0.000	0.333
24	0.333	0.000	0.333
25	0.333	0.000	0.333
26	0.222	0.000	0.222
27	0.222	0.000	0.222
28	0.222	0.000	0.222
29	0.222	0.000	0.222

\* this node already tested



#### 4.5.1 Calculating ProbMisused(s)

The probability that a node will be Misused ("ProbMisused") is:

$$\text{ProbMisused}(s) = \frac{\text{ElimMisused}(s)}{\text{ElimMastered}(s) + \text{ElimMisused}(s)}$$

$$\text{ProbMastered}(s) = \frac{\text{ElimMastered}(s)}{\text{ElimMastered}(s) + \text{ElimMisused}(s)}$$

This satisfies the condition that:

$$\text{ProbMisused}(s) + \text{ProbMastered}(s) = 1$$

We therefore have for ElimAve(s):

$$\begin{aligned} \text{ElimAve}(s) &= \text{ProbMastered}(s) \times \text{ElimMastered}(s) + \text{ProbMisused}(s) \times \text{ElimMisused}(s) \\ &= \frac{\text{ElimMastered}(s)^2}{\text{ElimMastered}(s) + \text{ElimMisused}(s)} \\ &+ \frac{\text{ElimMisused}(s)^2}{\text{ElimMastered}(s) + \text{ElimMisused}(s)} \\ &= \frac{\text{ElimMastered}(s)^2 + \text{ElimMisused}(s)^2}{\text{ElimMastered}(s) + \text{ElimMisused}(s)} \end{aligned}$$

Using the figures from Table 3 for ElimMastered(s) and ElimMisused(s), we can calculate the ElimAve(s) terms.

Caution: For nodes which have already been determined to be MISUSED or MASTERED, the denominator "ElimMasteredGood(s) + ElimMiused(s)" will evaluate to zero, forcing undefined division by zero. For these cases, set ElimAve(s) to "0".

#### 4.6 Final Selection

Table 4. Listing of ElimAve(s) Terms

Node s	ElimAve(s)
1	1.000
2	0.802
3	0.654
4	0.555
5	0.506
6	0.506
7	0.555
8	0.654
9	0.802
10	0.000 *
11	0.333
12	0.333
13	0.333
14	0.555
15	0.555
16	0.555
17	0.555
18	0.555
19	0.555
20	0.555
21	0.555
22	0.333
23	0.333
24	0.333
25	0.333
26	0.222
27	0.222
28	0.222
29	0.222

\* node already tested during a  
previous part task or whole task.

4. Determining the Student Model Node to Test

Table 5. Ranked listing of Nodes

Node sRank		Grouping
1	1.000	1 This is the node to test
2	0.802	2
9	0.802	
3	0.654	3
8	0.654	
4	0.555	4
7	0.555	
14	0.555	
15	0.555	
16	0.555	
17	0.555	
18	0.555	
19	0.555	
20	0.555	
21	0.555	
5	0.506	5
6	0.506	
11	0.333	6
12	0.333	
13	0.333	
22	0.333	
23	0.333	
24	0.333	
25	0.333	
26	0.222	7
27	0.222	
28	0.222	
29	0.222	
10	0	Unselectable (already measured)

